



# SAS<sup>®</sup> Viya<sup>™</sup> 3.1 ステートメント: リファレンス

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. SAS® *Viya*™ 3.1 ステートメント: リファレンス. Cary, NC: SAS Institute Inc.

**SAS® *Viya*™ 3.1 ステートメント: リファレンス**

Copyright © 2016, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

**For a hard copy book:** No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

**For a web download or e-book:** Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

**U.S. Government License Rights; Restricted Rights:** The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

September 2016

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

3.1-P1:lestmtsref

---

## 目次

<b>1 章・SAS ステートメント</b> .....	<b>1</b>
ステートメントについて .....	1
DATA ステップステートメント .....	1
グローバルステートメント .....	3
他の SAS ドキュメントで説明されているステートメント .....	3
<b>2 章・SAS ステートメントのディクショナリ</b> .....	<b>5</b>
カテゴリ別の DATA ステップステートメント .....	7
カテゴリ別のグローバルステートメント .....	14
ディクショナリ .....	17
<b>推奨資料</b> .....	<b>425</b>
<b>キーワード</b> .....	<b>427</b>



# 1 章

## SAS ステートメント

---

ステートメントについて .....	1
DATA ステップステートメント .....	1
実行ステートメントと宣言ステートメント .....	1
グローバルステートメント .....	3
他の SAS ドキュメントで説明されているステートメント .....	3

---

### ステートメントについて

SAS ステートメントとは、SAS キーワード、SAS 名、特殊文字、演算子で構成される文字列です。SAS キーワード、SAS 名、特殊文字、演算子で構成される文字列です。SAS ステートメントは、特定の処理を実行するよう SAS に指示するほか、必要な情報を SAS に提供します。SAS ステートメントの最後には、必ずセミコロン(;)を追加します。

本書では、次の 2 種類の SAS ステートメントについて説明します。

- DATA ステッププログラムで使用するステートメント
  - グローバルに適用され、SAS プログラムの任意の位置で使用できるステートメント
- 

### DATA ステップステートメント

#### 実行ステートメントと宣言ステートメント

DATA ステップステートメントとは、DATA ステートメント内に現れる実行または宣言ステートメントのことです。**実行ステートメント**は、DATA ステップの各繰り返し時に、何らかのアクションを実施します。一方、**宣言ステートメント**は、SAS に情報を供給するものであり、システムがプログラムをコンパイルする際に有効となります。

次の表に、CAS および SASDATA ステップで使用可能な実行ステートメントと宣言ステートメントを示します。

表 1.1 CAS および SASDATA ステップの実行ステートメント

実行ステートメント		
配列参照	IF、サブセット化	PUT (名前付き出力)
割り当て	IF-THEN/ELSE	PUT
CALL	LEAVE	PUT (ODS)
CONTINUE	LINK	PUTLOG
DELETE	LIST	RETURN
DO	LOSTCARD	SELECT
DO (反復)	MERGE	SET
DO UNTIL	OUTPUT	STOP
DO WHILE	PUT (カラム出力)	合計
ERROR	PUT (フォーマット出力)	
FILE	PUT (リスト出力)	

表 1.2 CAS および SASDATA ステップの宣言ステートメント

宣言ステートメント		
ARRAY	END	ラベル、ステートメント
BY	FORMAT	LENGTH
DATA	INFORMAT	RENAME
DROP	KEEP	RETAIN

次の表に、SASDATA ステップで使用可能な実行ステートメントと宣言ステートメントを示します。

表 1.3 SAS DATA ステップの実行ステートメント

実行ステートメント		
ABORT	INPUT (カラム入力)	REDIRECT
DESCRIBE	INPUT (フォーマット入力)	REMOVE
EXECUTE	INPUT (リスト入力)	REPLACE

実行ステートメント		
FILE (ODS)	INPUT (名前付き入力)	RESETLINE
GO TO	MODIFY	UPDATE
INFILE	ヌル	
INPUT	OUTPUT	

表 1.4 SAS DATA ステップの宣言ステートメント

宣言ステートメント		
ATTRIB	DATALINES	WHERE
CARDS	DATALINES4	
CARDS4	LABEL	

DATA ステップステートメントは、5つの機能カテゴリに分類することができます。カテゴリ別の SASDATA ステップステートメントの一覧については、“[カテゴリ別の DATA ステップステートメント](#)” (7 ページ) を参照してください。

## グローバルステートメント

グローバルステートメントは、通常、SAS への情報の提供、情報やデータの要求、異なる実行モード間での移行、またはシステムオプションの値の設定を行います。グローバルステートメントは SAS プログラムのどの位置にでも指定できます。グローバルステートメントは実行ステートメントではありません。また、SAS がプログラムステートメントをコンパイルした直後に有効になります。

グローバルステートメントは、7つの機能カテゴリに分類することができます。カテゴリ別のグローバルステップステートメントの一覧については、“[カテゴリ別のグローバルステートメント](#)” (14 ページ) を参照してください。

他の SAS ソフトウェアプロダクトには、それぞれのプロダクトで使用できるグローバルステートメントが用意されています。詳細については、各プロダクトの SAS マニュアルを参照してください。

## 他の SAS ドキュメントで説明されているステートメント

SAS Viya ステートメント: リファレンスに記載されているステートメントの他に、次のマニュアルでもステートメントについて説明しています。

- [SAS Viya DS2 Language Reference](#)
- [SAS Viya Macro Language: Reference](#)

#### 4 1章 • SAS ステートメント

- *SAS Viya Data Management and Utility Procedures Guide*
- *SAS/CONNECT for SAS Viya: User's Guide*



## 2 章

SAS ステートメントのディクショ  
ナリ

---

カテゴリ別の DATA ステップステートメント .....	7
カテゴリ別のグローバルステートメント .....	14
ディクショナリ .....	17
ABORT ステートメント .....	17
ARRAY ステートメント .....	20
配列参照ステートメント .....	25
割り当てステートメント .....	28
ATTRIB ステートメント .....	29
BY ステートメント .....	33
CALL ステートメント .....	39
CARDS ステートメント .....	39
CARDS4 ステートメント .....	40
CATNAME ステートメント .....	40
CHECKPOINT EXECUTE_ALWAYS ステートメント .....	43
コメントステートメント .....	44
CONTINUE ステートメント .....	46
DATA ステートメント .....	47
DATALINES ステートメント .....	57
DATALINES4 ステートメント .....	59
DELETE ステートメント .....	60
DESCRIBE ステートメント .....	61
DO ステートメント .....	62
DO ステートメント、反復 .....	63
DO UNTIL ステートメント .....	67
DO WHILE ステートメント .....	69
DROP ステートメント .....	70
END ステートメント .....	72
ENDSAS ステートメント .....	72
ERROR ステートメント .....	73
EXECUTE ステートメント .....	74
FILE ステートメント .....	75
FILENAME ステートメント .....	96
FILENAME ステートメント、SFTP アクセス方式 .....	109
FILENAME ステートメント、URL アクセス方式 .....	115
FILENAME ステートメント、ZIP アクセス方式 .....	121
FOOTNOTE ステートメント .....	125
FORMAT ステートメント .....	129
GO TO ステートメント .....	132
IF ステートメント、サブセット化 .....	134
IF-THEN/ELSE ステートメント .....	137

%INCLUDE ステートメント	138
INFILE ステートメント	146
INFORMAT ステートメント	176
INPUT ステートメント	179
INPUT ステートメント、カラム	197
INPUT ステートメント、フォーマット	200
INPUT ステートメント、リスト	204
INPUT ステートメント、名前付き	212
KEEP ステートメント	216
LABEL ステートメント	217
ラベル: ステートメント	219
LEAVE ステートメント	221
LENGTH ステートメント	222
LIBNAME ステートメント	225
LIBNAME ステートメント, CVP Engine	243
LIBNAME ステートメント, JMP Engine	247
LIBNAME ステートメント, JSON Engine	249
LIBNAME ステートメント, WebDAV サーバーアクセス	262
LINK ステートメント	266
LIST ステートメント	268
%LIST ステートメント	270
LOCK ステートメント	271
LOCKDOWN ステートメント	274
LOSTCARD ステートメント	278
MERGE ステートメント	280
MISSING ステートメント	288
MODIFY ステートメント	289
ヌルステートメント	311
OPTIONS ステートメント	313
OUTPUT ステートメント	314
PAGE ステートメント	317
PUT ステートメント	318
PUT ステートメント、カラム	336
PUT ステートメント、フォーマット	338
PUT ステートメント、リスト	343
PUT ステートメント、名前付き	347
PUTLOG ステートメント	350
REDIRECT ステートメント	352
REMOVE ステートメント	354
RENAME ステートメント	356
REPLACE ステートメント	357
RESETLINE ステートメント	360
RETAIN ステートメント	361
RETURN ステートメント	365
RUN ステートメント	367
%RUN ステートメント	368
SASFILE ステートメント	369
SELECT ステートメント	375
SET ステートメント	378
SKIP ステートメント	392
STOP ステートメント	393
合計ステートメント	394
SYSECHO ステートメント	395
SYSTASK ステートメント	396
TITLE ステートメント	399
UPDATE ステートメント	408

WAITFOR ステートメント ..... 415  
 WHERE ステートメント ..... 416  
 X ステートメント ..... 423

## カテゴリ別の DATA ステップステートメント

実行ステートメントと宣言ステートメントの他に、CAS 用の SAS DATA ステップステートメントは 5 つの機能カテゴリに分類できます。

表 2.1 DATA ステップステートメントのカテゴリ

ステートメントのカテゴリ	機能
アクション	<ul style="list-style-type: none"> <li>変数の作成や変更を行います。</li> <li>DATA ステップで処理する特定のオブザベーションのみを選択します。</li> <li>入力データのエラーを調べます。</li> <li>作成されるオブザベーションを操作します。</li> </ul>
CAS	<ul style="list-style-type: none"> <li>CAS サーバーで実行されるステートメント</li> </ul>
制御	<ul style="list-style-type: none"> <li>特定のオブザベーションに対して、ステートメントをスキップします。</li> <li>ステートメントの実行順序を変更します。</li> <li>ある箇所から別の箇所にプログラムの制御を移動します。</li> </ul>
ファイル操作	<ul style="list-style-type: none"> <li>データセットへの入力として使用するファイルを操作します。</li> <li>DATA ステップで書き込むファイルを操作します。</li> </ul>
情報	<ul style="list-style-type: none"> <li>プログラムデータベクトルに関する追加情報を提供します。</li> <li>作成する 1 つまたは複数のデータセットに関する追加情報を提供します。</li> </ul>

次の表に、カテゴリ別の DATA ステップステートメントの一覧と簡単な説明を示します。

カテゴリ	言語要素	説明
CAS	ARRAY ステートメント (p. 20)	配列の要素を定義します。
	配列参照ステートメント (p. 25)	処理する配列要素を記述します。

カテゴリ	言語要素	説明
	割り当てステートメント (p. 28)	式を評価し、その結果を変数に格納します。
	BY ステートメント (p. 33)	DATA ステップ内の SET、MERGE、MODIFY、UPDATE の各ステートメントの実行を制御し、特別なグループ変数を設定します。
	CALL ステートメント (p. 39)	SAS CALL ルーチンを呼び出します。
	CONTINUE ステートメント (p. 46)	DO ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。
	DATA ステートメント (p. 47)	DATA ステップを開始します。また、出力データセット、ビュー、プログラムの名前を指定します。
	DELETE ステートメント (p. 60)	現在のオブザベーションの処理を中止します。
	DO ステートメント (p. 62)	1 単位として実行するステートメントのグループを指定します。
	DO ステートメント、反復 (p. 63)	インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。
	DO UNTIL ステートメント (p. 67)	条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。
	DO WHILE ステートメント (p. 69)	条件が真の間、DO ループ内のステートメントを繰り返し実行します。
	DROP ステートメント (p. 70)	出力 SAS データセットから変数を除外します。
	END ステートメント (p. 72)	DO グループまたは SELECT グループの処理を終了します。
	ERROR ステートメント (p. 73)	_ERROR_ を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。
	FILE ステートメント (p. 75)	PUT ステートメントの現在の出力ファイルを指定します。
	FORMAT ステートメント (p. 129)	変数に出力形式を関連付けます。
	GO TO ステートメント (p. 132)	指定したステートメントラベルにプログラム実行を移動します。RETURN ステートメントが後に続く場合、DATA ステップの先頭に実行を戻します。
	IF ステートメント、サブセット化 (p. 134)	指定した式の条件に一致するオブザベーションの処理を継続します。

カテゴリ	言語要素	説明
	IF-THEN/ELSE ステートメント (p. 137)	特定の条件を満たすオブザベーションに対して、SAS ステートメントを実行します。
	INFORMAT ステートメント (p. 176)	変数に入力形式を関連付けます。
	KEEP ステートメント (p. 216)	出力 SAS データセットに含める変数を指定します。
	LABEL ステートメント (p. 217)	説明を示すラベルを変数に割り当てます。
	ラベル: ステートメント (p. 219)	他のステートメントから参照されるステートメントを識別します。
	LEAVE ステートメント (p. 221)	現在のループの処理を中止し、シーケンス内の次のステートメントから再開します。
	LENGTH ステートメント (p. 222)	変数の保存時に使用するバイト数を指定します。
	LINK ステートメント (p. 266)	指定したステートメントラベルにプログラム実行移動します。RETURN ステートメントが後に続く場合、LINK ステートメントの直後のステートメントに実行を戻します。
	LIST ステートメント (p. 268)	現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。
	LOSTCARD ステートメント (p. 278)	1 オブザベーションあたりのレコードが複数あるデータに欠損レコードや無効なレコードが検出された際に、入力データを再同期します。
	MERGE ステートメント (p. 280)	複数の SAS データセットにある複数のオブザベーションを 1 つのオブザベーションに結合します。
	ヌルステートメント (p. 311)	データの終わりを示すか、またはプレースホルダとして動作します。
	OUTPUT ステートメント (p. 314)	現在のオブザベーションを SAS データセットに書き込みます。
	PUT ステートメント (p. 318)	SAS ログ、結果ウィンドウ、または最後の FILE ステートメントに指定した外部の場所に行を出力します。
	PUT ステートメント、カラム (p. 336)	変数の値を出力行の指定した列に書き込みます。
	PUT ステートメント、フォーマット (p. 338)	指定した出力形式で変数の値を出力行に書き込みます。
	PUT ステートメント、リスト (p. 343)	変数値と指定した文字列を出力行に書き込みます。

カテゴリ	言語要素	説明
	PUT ステートメント、名前付き (p. 347)	変数名と等号の後に変数値を書き込みます。
	PUTLOG ステートメント (p. 350)	SAS ログにメッセージを書き込みます。
	RENAME ステートメント (p. 356)	出力 SAS データセットの変数に新しい名前を指定します。
	RETAIN ステートメント (p. 361)	DATA ステップを繰り返す際に、INPUT ステートメントまたは割り当てステートメントで作成される変数の値を保持します。
	RETURN ステートメント (p. 365)	DATA ステップ内の現在の位置でステートメントの実行を中止し、ステップで事前に定義されている位置に戻ります。
	SELECT ステートメント (p. 375)	複数のステートメントまたはステートメントグループからその 1 つを実行します。
	SET ステートメント (p. 378)	1 つ以上の SAS データセットからオブザベーションを読み込みます。
	STOP ステートメント (p. 393)	現在の DATA ステップの実行を中止します。
	合計ステートメント (p. 394)	式の計算結果を合計変数に加算します。
アクション	ABORT ステートメント (p. 17)	現在の DATA ステップ、SAS ジョブまたは SAS セッションの実行を中止します。
	割り当てステートメント (p. 28)	式を評価し、その結果を変数に格納します。
	CALL ステートメント (p. 39)	SAS CALL ルーチンを呼び出します。
	DELETE ステートメント (p. 60)	現在のオブザベーションの処理を中止します。
	DESCRIBE ステートメント (p. 61)	コンパイル済み DATA ステッププログラムまたは DATA ステップビューからソースコードを読み込みます。
	ERROR ステートメント (p. 73)	_ERROR_ を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。
	EXECUTE ステートメント (p. 74)	コンパイル済み DATA ステッププログラムを実行します。
	IF ステートメント、サブセット化 (p. 134)	指定した式の条件に一致するオブザベーションの処理を継続します。
	LIST ステートメント (p. 268)	現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。

カテゴリ	言語要素	説明
	LOSTCARD ステートメント (p. 278)	1 オブザベーションあたりのレコードが複数あるデータに欠損レコードや無効なレコードが検出された際に、入力データを再同期します。
	ヌルステートメント (p. 311)	データの終わりを示すか、またはブレースホルダとして動作します。
	OUTPUT ステートメント (p. 314)	現在のオブザベーションを SAS データセットに書き込みます。
	PUTLOG ステートメント (p. 350)	SAS ログにメッセージを書き込みます。
	REDIRECT ステートメント (p. 352)	ストアドプログラムを実行するときに、別の入力 SAS データセットまたは出力 SAS データセットを参照します。
	REMOVE ステートメント (p. 354)	SAS データセットからオブザベーションを削除します。
	REPLACE ステートメント (p. 357)	同じ場所にあるオブザベーションを置換します。
	STOP ステートメント (p. 393)	現在の DATA ステップの実行を中止します。
	合計ステートメント (p. 394)	式の計算結果を合計変数に加算します。
	WHERE ステートメント (p. 416)	SAS データセットから特定の条件を満たすオブザベーションを選択します。
情報	ARRAY ステートメント (p. 20)	配列の要素を定義します。
	配列参照ステートメント (p. 25)	処理する配列要素を記述します。
	ATTRIB ステートメント (p. 29)	1 つまたは複数の変数に出力形式、入力形式、ラベル、長さを設定します。
	DROP ステートメント (p. 70)	出力 SAS データセットから変数を除外します。
	FORMAT ステートメント (p. 129)	変数に出力形式を関連付けます。
	INFORMAT ステートメント (p. 176)	変数に入力形式を関連付けます。
	KEEP ステートメント (p. 216)	出力 SAS データセットに含める変数を指定します。
	LABEL ステートメント (p. 217)	説明を示すラベルを変数に割り当てます。

カテゴリ	言語要素	説明
	LENGTH ステートメント (p. 222)	変数の保存時に使用するバイト数を指定します。
	RENAME ステートメント (p. 356)	出力 SAS データセットの変数に新しい名前を指定します。
	RETAIN ステートメント (p. 361)	DATA ステップを繰り返す際に、INPUT ステートメントまたは割り当てステートメントで作成される変数の値を保持します。
制御	CONTINUE ステートメント (p. 46)	DO ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。
	DO ステートメント (p. 62)	1 単位として実行するステートメントのグループを指定します。
	DO ステートメント、反復 (p. 63)	インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。
	DO UNTIL ステートメント (p. 67)	条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。
	DO WHILE ステートメント (p. 69)	条件が真の間、DO ループ内のステートメントを繰り返し実行します。
	END ステートメント (p. 72)	DO グループまたは SELECT グループの処理を終了します。
	GO TO ステートメント (p. 132)	指定したステートメントラベルにプログラム実行を移動します。RETURN ステートメントが後に続く場合、DATA ステップの先頭に戻ります。
	IF-THEN/ELSE ステートメント (p. 137)	特定の条件を満たすオブザベーションに対して、SAS ステートメントを実行します。
	ラベル: ステートメント (p. 219)	他のステートメントから参照されるステートメントを識別します。
	LEAVE ステートメント (p. 221)	現在のループの処理を中止し、シーケンス内の次のステートメントから再開します。
	LINK ステートメント (p. 266)	指定したステートメントラベルにプログラム実行移動します。RETURN ステートメントが後に続く場合、LINK ステートメントの直後のステートメントに戻ります。
	RETURN ステートメント (p. 365)	DATA ステップ内の現在の位置でステートメントの実行を中止し、ステップで事前に定義されている位置に戻ります。
	SELECT ステートメント (p. 375)	複数のステートメントまたはステートメントグループからその 1 つを実行します。



カテゴリ	言語要素	説明
ファイル操作	BY ステートメント (p. 33)	DATA ステップ内の SET、MERGE、MODIFY、UPDATE の各ステートメントの実行を制御し、特別なグループ変数を設定します。
	CARDS ステートメント (p. 39)	データ行の始まりを示します。
	CARDS4 ステートメント (p. 40)	セミコロンが含まれるデータ行の始まりを示します。
	DATA ステートメント (p. 47)	DATA ステップを開始します。また、出力データセット、ビュー、プログラムの名前を指定します。
	DATALINES ステートメント (p. 57)	データ行の始まりを示します。
	DATALINES4 ステートメント (p. 59)	セミコロンが含まれるデータ行の始まりを指定します。
	FILE ステートメント (p. 75)	PUT ステートメントの現在の出力ファイルを指定します。
	INFILE ステートメント (p. 146)	INPUT ステートメントで読み込む外部ファイルを指定します。
	INPUT ステートメント (p. 179)	入力データレコードでの値の位置を指定し、読み込んだ値に対応する SAS 変数に割り当てます。
	INPUT ステートメント、カラム (p. 197)	指定した列から入力値を読み込み、読み込んだ値に対応する SAS 変数に割り当てます。
	INPUT ステートメント、フォーマット (p. 200)	指定した形式で入力値を読み込み、読み込んだ値に対応する SAS 変数に割り当てます。
	INPUT ステートメント、リスト (p. 204)	入力データレコードの入力値を走査し、対応する SAS 変数に割り当てます。
	INPUT ステートメント、名前付き (p. 212)	変数名と等号の後ろに続くデータ値を読み込み、読み込んだ値に対応する SAS 変数に割り当てます。
	MERGE ステートメント (p. 280)	複数の SAS データセットにある複数のオブザベーションを 1 つのオブザベーションに結合します。
	MODIFY ステートメント (p. 289)	既存のデータセットにあるオブザベーションの置き換え、削除、追加を実行します。ただし、オブザベーションのコピーは作成しません。
PUT ステートメント (p. 318)	SAS ログ、結果ウィンドウ、または最後の FILE ステートメントに指定した外部の場所に行を出力します。	
PUT ステートメント、カラム (p. 336)	変数の値を出力行の指定した列に書き込みます。	

カテゴリ	言語要素	説明
	PUT ステートメント、フォーマット	指定した出力形式で変数の値を出力行に書き込みます。 (p. 338)
	PUT ステートメント、リスト	変数値と指定した文字列を出力行に書き込みます。 (p. 343)
	PUT ステートメント、名前付き	変数名と等号の後に変数値を書き込みます。 (p. 347)
	SET ステートメント (p. 378)	1 つ以上の SAS データセットからオブザベーションを読み込みます。
	UPDATE ステートメント (p. 408)	トランザクションを適用してマスタファイルを更新します。

## カテゴリ別のグローバルステートメント

次の表に、機能別に 7 つのカテゴリにまとめた SAS グローバルステートメントの一覧と簡単な説明を示します。

表 2.2 カテゴリ別のグローバルステートメント

ステートメントのカテゴリ	機能
CAS	CAS サーバーで実行されるステートメント
データアクセス	SAS ライブラリ、SAS カタログ、外部ファイル、出力デバイスに参照名を関連付け、リモートファイルにアクセスします。
情報	プログラムデータベクトルに関する追加情報を提供します。
ログ制御	SAS ログの出力内容を変更します。
動作環境	動作環境に直接アクセスします。
出力制御	SAS 出力にタイトルとフットノートを追加します。また、さまざまな形式で出力を提供します。
プログラム制御	SAS プログラムを処理する方法を制御します。

次の表には、SAS グローバルステートメントの一覧と簡単な説明を示します。詳細については、各ステートメントの説明を参照してください。

カテゴリ	言語要素	説明
CAS	コメントステートメント (p. 44)	ステートメントまたはプログラムの目的を説明します。
	RUN ステートメント (p. 367)	すでに入力されている SAS ステートメントを実行します。
出力制御	FOOTNOTE ステートメント (p. 125)	プロシジャまたは DATA ステップの出力の下部にテキストを 10 行まで書き込みます。
	TITLE ステートメント (p. 399)	SAS 出力に使用するタイトルを指定します。
情報	MISSING ステートメント (p. 288)	入力データ内で数値データの特殊欠損値を表す文字を割り当てます。
制御	LOCKDOWN ステートメント (p. 274)	サーバープロセスの内部からホスト動作環境へのアクセスを制限することにより、ワークスペースまたは SAS/CONNECT サーバーを保護します。
データアクセス	CATNAME ステートメント (p. 40)	複数のカタログをカタログ参照名(ショートカット名)に関連付け、1つのカタログに論理的に結合します。また、1つまたはすべてのカタログ参照名をクリアします。1つの連結またはすべての連結に含まれる連結カタログをリストします。
	FILENAME ステートメント (p. 96)	SAS ファイル参照名を外部ファイルまたは出力デバイスと関連付けたり、ファイル参照名と外部ファイルの関連付けを取り消します。また、外部ファイルの属性を書き込みます。
	FILENAME ステートメント、SFTP アクセス方式 (p. 109)	SFTP プロトコルを使用してリモートファイルにアクセスできます。
	FILENAME ステートメント、URL アクセス方式 (p. 115)	URL アクセス方式を使用してリモートファイルにアクセスできます。
	FILENAME ステートメント、ZIP アクセス方式 (p. 121)	ZIP ファイルにアクセスできるようにします。
	LIBNAME ステートメント (p. 225)	SAS ライブラリへのライブラリ参照名(ショートカット名)の関連付けや関連付けの取り消し、1つまたはすべてのライブラリ参照名の関連付けの取り消しを実行します。また、SAS ライブラリの属性のリスト、SAS ライブラリの連結や SAS カタログの連結を実行します。
	LIBNAME ステートメント、CVP Engine (p. 243)	ファイルのトランスコードが必要なときに文字データの切り捨てが発生しなくなるように、文字変数の長さを拡張する文字変数埋め込み(CVP)エンジンのライブラリ参照名を関連付けます。
	LIBNAME ステートメント、JMP Engine (p. 247)	ライブラリ参照名を JMP データテーブルに関連付け、JMP データテーブルの読み込みや書き込みを実行できるようにします。

カテゴリ	言語要素	説明
	LIBNAME ステートメント, JSON Engine (p. 249)	SAS ライブラリ参照名を JSON ドキュメントと関連付けて、JSON データと、指定された任意の MAP が有効になることを保証します。
	LIBNAME ステートメント, WebDAV サーバーアクセス (p. 262)	SAS ライブラリにライブラリ参照名を関連付け、WebDAV (Web-based Distributed Authoring And Versioning) サーバーにアクセスできるようにします。
動作環境	SYSTASK ステートメント (p. 396)	非同期タスクを一覧表示します。
	WAITFOR ステートメント (p. 415)	タスクの実行が終了するまで、現在の SAS セッションの実行を中断します。
	X ステートメント (p. 423)	SAS セッションから動作環境のコマンドを発行します。
プログラム制御	CHECKPOINT EXECUTE_ALWAYS ステートメント (p. 43)	チェックポイント-再開データを無視して、このステートメントの直後にある DATA ステップまたは PROC ステップを実行するように指示します。
	ENDSAS ステートメント (p. 72)	現在の DATA ステップまたは PROC ステップを実行した後、SAS ジョブまたは SAS セッションを終了します。
	%INCLUDE ステートメント (p. 138)	SAS プログラミングステートメント、データ行、またはその両方を現在の SAS プログラムに読み込みます。
	%LIST ステートメント (p. 270)	現在のセッション中に入力された行を表示します。
	LOCK ステートメント (p. 271)	既存の SAS ファイルに対する排他的ロックの取得、リスト出力、または解除を行います。
	OPTIONS ステートメント (p. 313)	1 つまたは複数の SAS システムオプションの値の指定や変更を行います。
	RUN ステートメント (p. 367)	すでに入力されている SAS ステートメントを実行します。
	%RUN ステートメント (p. 368)	%INCLUDE *ステートメントの後にあるソースステートメントを終了します。
	SASFILE ステートメント (p. 369)	SAS データセットを開き、ファイル全体をメモリに保持するために必要なバッファを割り当てます。
	SYSECHO ステートメント (p. 395)	グローバルステートメントの完了イベントを送信し、テキスト文字列を IOM クライアントに戻します。
ログ制御	コメントステートメント (p. 44)	ステートメントまたはプログラムの目的を説明します。
	PAGE ステートメント (p. 317)	SAS ログの新しいページにスキップします。

カテゴリ	言語要素	説明
	RESETLINE ステートメント (p. 360)	SAS ログ内のプログラムの行番号を 1 にリセットします。
	SKIP ステートメント (p. 392)	SAS ログ内に空白行を 1 行作成します。

## ディクショナリ

### ABORT ステートメント

現在の DATA ステップ、SAS ジョブまたは SAS セッションの実行を中止します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。

#### 構文

**ABORT** <ABEND | RETURN> <n>;

#### 引数なし

引数を指定しない場合、ABORT ステートメントは実行モードに応じて、次のような結果をもたらします。

バッチモードおよび非対話型モード

- 現在の DATA ステップを終了し、SAS ログにエラーメッセージを書き込みます。ABORT ステートメントを実行するタイミングによって、データセットの一部または全部のオブザベーションが含まれない場合があります。
- OBS=システムオプションを 0 に設定します。
- SAS ジョブの残りの部分に関して、マクロステートメントの実行、システムオプションステートメントの実行、プログラムステートメントの構文チェックなどの限られた処理を続けます。
- 後続の DATA ステップおよび PROC ステップには、オブザベーションを含まない出力データセットを作成します。

SAS Studio

- 現在処理中の DATA ステップを終了します。
- ABORT ステートメントの実行前に処理されたオブザベーションを含むデータセットが作成されます。
- ABORT ステートメントによる DATA ステップの終了を示すメッセージをログに書き込みます。

- ABORT ステートメントの後続の DATA ステップまたは PROC ステートメントの処理を続けます。

#### 対話型ラインモード

現在処理中の DATA ステップを終了します。後続の DATA ステップまたは PROC ステートメントは、通常と同様に処理されます。

## 引数

### ABEND

現在実行中の SAS ジョブまたは SAS セッションを異常終了させます。結果は実行モードによって異なります。

- バッチモードおよび非対話型モード
  - 現在実行中の処理をすぐに終了します。
  - ABORT ステートメントの ABEND オプションにより実行が終了させられたことを示すエラーメッセージを SAS ログに送信します。
  - 後続のステートメントや構文チェックを実行しません。
  - 動作環境に制御を戻します。それ以降の動作は、動作環境と各サイトでの異常終了時のジョブの処理方法によって異なります。
- SAS Studio および対話型ラインモード
  - 現在実行中の処理をすぐに中止し、動作環境にユーザーを戻します。

### RETURN

現在実行中の SAS ジョブまたはセッションをすぐに通常終了します。結果は実行モードによって異なります。

- バッチモードおよび非対話型モード
  - 現在実行中の処理をすぐに終了します。
  - ABORT ステートメントの RETURN オプションによる処理の終了を示すエラーメッセージをログに書き込みます。
  - 後続のステートメントや構文チェックは実行しません。
  - エラーを示す条件コードが表示され、動作環境に制御を戻します。
- SAS Studio
  - 現在実行中の処理をすぐに中止し、動作環境にユーザーを戻します。

### *n*

終了ステータスコードを表す整数値で、SAS が処理を終了してシェルに戻る際、この値を返します。*n* の値は 0 - 255 の範囲で指定できます。通常、エラーがなかった場合は、リターンコードとして 0 を返します。正值であれば、数値が大きいほど重大なエラーであることを表します。0 - 6 のリターンコードと 977 より大きいコードは、SAS での使用のために予約されます。

### NOLIST

すべての変数を SAS ログへ出力しないようにします。

**要件** ABORT ステートメントでは NOLIST を最後のオプションとして指定する必要があります。

## 詳細

### 概要情報

ABORT ステートメントは、現在の DATA ステップの処理を終了させます。後続の処理は、次によって異なります。

- SAS ステートメントをサブミットするために使用した方法
- ABORT ステートメントで使用した引数
- 使用中の動作環境

通常、ABORT ステートメントは IF-THEN ステートメントや SELECT ステートメントの句の中で使用され、エラー条件が発生した場合に処理を終了させます。

注: システムオプション ERRORABEND が適用されている場合、SAS では ABORT ステートメントで生成されるリターンコードは無視されます。

注: DATA ステップで ABORT ステートメントを実行する場合、このステップで作成された DATA ステップで同じ名前の既存のデータセットが置き換えられることはありません。

### 動作環境の情報

ABEND オプションと RETURN オプションには 1 つだけ違いがあります。ABEND オプションを使用した場合、それ以降の処理は、お使いの動作環境と各サイトでの異常終了ジョブの処理方法によって異なります。RETURN オプションを使用した場合、エラーを示す条件コードが返されます。

## 比較

- SAS Studio または対話型ラインモードを使用する場合、ABORT ステートメントと STOP ステートメントの両方とも処理を中止します。ABORT ステートメントでは自動変数\_ERROR\_の値を 1 に設定しますが、STOP ステートメントでは設定しません。
- バッチモードまたは非対話型モードでは、ABORT ステートメントと STOP ステートメントの動作が異なります。どちらのステートメントも処理を中止しますが、ABORT ステートメントのみ自動変数\_ERROR\_を 1 に設定します。STOP ステートメントを使用すると、現在実行中の DATA ステップのみを中止し、次のステップから処理を続けます。

## 例: SAS の実行中止

この例では、IF-THEN ステートメント内で ABORT ステートメントを使用しており、ゼロ除算の条件が発生するデータ値が検出された場合に SAS の処理を中止します。

```
if volume=0 then abort 255;
  density=mass/volume;
```

ABORT ステートメントの実行時、*n* 値に指定された条件コード 255 が動作環境に返されます。

## 関連項目:

- [“Determining the Completion Status of a SAS Job in Linux Environments” \(Batch and Line Mode Processing in SAS Viya\)](#)

## ステートメント:

- “STOP ステートメント” (393 ページ)

---

## ARRAY ステートメント

配列の要素を定義します。

該当要素:	DATA ステップ
カテゴリ:	CAS 情報
種類:	宣言
制限事項:	配列の初期化時に、VARCHAR データタイプの変数をサポートします。

---

### 構文

```
ARRAY array-name { subscript } <$> <<VARCHAR (length | **)> | length>
<array-elements> <(initial-value-list)>;
```

### 引数

#### *array-name*

配列の名前を指定します。

**制限事項** *Array-name* には、同じ DATA ステップ内の SAS 変数名とは異なる SAS 名を指定してください。

**注意** SAS 関数の名前を配列名として使用すると、予期しない結果が生じることがあります。関数名を誤って配列名として使用すると、DATA ステップの実行時に、指定した名前を含むかっこ付きの参照名は関数の呼び出しではなく配列参照として解釈されます。この結果、警告メッセージが SAS ログに書き込まれます。

#### {*subscript*}

アスタリスク、数値、または数値の範囲を使用して、配列内の要素数と配置順序を指定します。*subscript* は、次のいずれかの形式になります。

#### {*dimension-size(s)*}

配列の各次元の要素数を指定します。*dimension-size* は、1 次元配列の要素数または多次元配列の各次元の要素数を示す数値です。

**ヒント** *subscript* は中かっこ({})、大かっこ([ ])、丸かっこ(( ))で囲むことができます。

**例** 次の ARRAY ステートメントは、SIMPLE という名前の 1 次元配列を定義します。SIMPLE 配列には、RED、GREEN、YELLOW という 3 つの変数がまとめられています。

```
array simple{3} red green yellow;
```

次元が複数ある配列は、多次元配列と呼ばれます。多次元配列の次元数に制限はありません。たとえば、2 次元配列は、配列要素の行列配置を提供します。SAS では変数を 2 次元配列に配置する場合、配列の上部左端から開始して、すべての行を順番に書き込みます(これを



「行順」と呼びます)。次のステートメントで、5 行 3 列で構成される 2 次元配列を定義します。

```
array x{5,3} score1-score15;
```

---

**{<lower :>upper<, ...<lower :> upper>}**

*lower* および *upper* は、それぞれ配列の各次元の上限と下限です。

**範** 多くの明示的な配列では、配列の各次元の subscript は 1 から *n* ま  
**囲** でになります。*n* は、この次元の要素数になります。

**ヒ** ほとんどの配列では 1 を下限とする方が便利であるため、上限およ  
**ン** び下限を指定する必要はありません。ただし、配列次元を 1 以外の  
**ト** 数値から開始する場合、上限値と下限値の両方を指定することをお  
勧めします。

subscript の評価に必要な時間を短縮するには、下限として 0 を指定  
します。

**例** 次の例では、各次元の値は、デフォルトで各次元の上限になります。  
array x{5,3} score1-score15;

次の ARRAY ステートメントでは、もう 1 つの方法として前の例を長い  
形式で示します。

```
array x{1:5,1:3} score1-score15;
```

---

**{\*}**

配列内の変数の数から subscript を算出するように指定します。アスタ  
リスクを指定する場合、*array-elements* も指定する必要があります。

**制限事** アスタリスクは、\_TEMPORARY\_ 配列では使用できないほか、多  
**項** 次元配列の定義時にも使用できません。

---

**\$**

配列内の要素が文字要素であることを指定します。

**ヒント** 配列要素が文字要素であることがすでに定義されている場合、ドル記  
号を指定する必要はありません。

## VARCHAR

直前にある変数が、VARCHAR データタイプの文字変数であることを指定し  
ます。

### *length*

配列要素の長さが設定されていない場合に配列要素の長さを指定します。

**範** CHAR 変数の場合、1 から 32767 まで VARCHAR 変数の場合、1 から 1  
**囲** - ((2\*\*31 - 1) / 4) まで数値変数の場合、3 から 8 まで数値変数に指定で  
きる長さの最小値は、ユーザーのシステムが使用する浮動小数点形式に  
よって異なります。ほとんどのシステムでは IEEE 浮動小数点形式が使  
用され、最小値は 3 バイトです。

---

**\*\***

許可されている最大長(現時点で 2\*\*31-1)を使用することを意味します。

### *array-elements*

配列を構成する要素の名前を指定します。*array-elements* は、すべて数値で  
指定するか、すべて文字で指定します。任意の順序で指定できます。これら  
の要素は次のとおりです。

**variables**

変数名をリストします。

**範囲** この名前は、ARRAY ステートメントにて定義した変数か、配列名と数値を組み合わせて SAS によって作成された変数のいずれかになります。たとえば、subscript がアスタリスクではなく数値である場合、配列内にある各変数の名前を指定する必要はありません。かわりに、配列名と数値 1、2、3、...*n* を組み合わせた変数名が SAS によって作成されます。

**制限事項** `_ALL_` を使用する場合、過去に定義済みの変数はすべて同一の種類である必要があります。

**ヒント** このような SAS 変数のリストにより、同一の DATA ステップで過去に定義済みの変数を参照できます。

`_NUMERIC_` は、すべての数値変数を示します。

`_CHARACTER_` は、すべての文字変数を示します。

`_ALL_` を使用すると、すべての変数を示します。

**例** “例 1: 配列を定義する” (24 ページ)

**`_TEMPORARY_`**

一時データ要素で構成されるリストを作成します。

**範囲** 一時データ要素は、数値または文字列になります。

**ヒント** 一時データ要素は DATA ステップの変数と同じように動作しますが、次の例外があります。

一時データ要素には名前がありません。一時データ要素の参照には、配列名と次元が使用されます。

出力データセットには書き出されません。

すべての要素を参照するために、特殊な subscript であるアスタリスク(\*)は使用できません。

一時データ要素の値は常に自動的に保持されます。この値は、DATA ステップの次の繰り返しの先頭に戻った際に欠損値にリセットされません。

配列を作成する目的が演算の実行のみにある場合、一時要素の配列を使用すると便利です。演算結果を保存するには、その結果を変数に割り当てます。一時データ要素を使用すると処理時間を短縮することができます。

**(initial-value-list)**

配列内の対応する要素に初期値を指定します。要素の値には、数値または文字列を指定します。文字列はすべて引用符で囲む必要があります。1 つまたは複数の初期値を直接指定するには、次の形式を使用します。

(initial-value(s))

繰り返しの回数とネスト形式のサブリストを初期値に指定するには、次の形式を使用します。

```
<constant-iter-value*> <( >constant value | constant-sublist< )>
```

**制限事項** *initial-value-list* と *array-elements* の両方を指定する場合、ARRAY ステートメントでは *array-elements* を *initial-value-list* より前に指定する必要があります。

**ヒント** 変数と一時データ要素の両方に初期値を割り当てることができます。

要素と値は位置を基準にして照合されます。指定した初期値の数よりも配列数が多い場合、初期値が割り当てられなかった配列要素には欠損値が割り当てられます。また、警告メッセージが発行されます。

初期値のリストはカンマまたはブランクを使用して区切ることができます。

連続する整数値の範囲を指定する場合は、簡略化した表記を使用できます。増分は常に+1 になります。

配列要素の属性(長さや種類など)を指定していない場合は、指定した初期値の属性が対応する配列要素に自動的に割り当てられます。初期値は、配列要素に新しい値が割り当てられるまで保持されます。

一部またはすべての要素に初期値を割り当てると、すべての要素は RETAIN ステートメントに指定した場合と同じように動作します。

**例** 次の例では、反復係数とネストされたサブリストを使用する方法を示します。次のすべての ARRAY ステートメントでは、同じ初期値リストが使用されています。

```
ARRAY x{10} x1-x10 (10*5);
ARRAY x{10} x1-x10 (5*(5 5));
ARRAY x{10} x1-x10 (5 5 3*(5 5) 5 5);
ARRAY x{10} x1-x10 (2*(5 5) 5 5 2*(5 5));
ARRAY x{10} x1-x10 (2*(5 2*(5 5)));
```

**例** [“例 2: 数値の初期値を割り当てる” \(24 ページ\)](#)

[“例 3: 文字列の初期値を割り当てる” \(24 ページ\)](#)

## 詳細

ARRAY ステートメントは、グループとして処理する要素のまとまりを定義します。配列の要素は、配列名と subscript を基準にして参照されます。1 つの配列内で複数の要素を処理する機会が多いため、配列は DO グループ内で頻繁に参照されます。

## 比較

- SAS 言語で使用する配列は、他の多くの言語で使用する配列とは異なります。SAS の配列は、変数のグループを一時的に識別するための便利な方法として使用します。この配列はデータ構造ではありません。また、*array-name* は変数ではありません。

- ARRAY ステートメントは、配列を定義します。配列参照は、プログラムステートメントにて配列要素を使用します。

## 例

### 例 1: 配列を定義する

- array rain {5} janr febr marr aprr mayr;
- array days{7} d1-d7;
- array month{\*} jan feb jul oct nov;
- array x{\*} \_NUMERIC\_;
- array qbx{10};
- array meal{3};

### 例 2: 数値の初期値を割り当てる

- array test{4} t1 t2 t3 t4 (90 80 70 70);
- array test{4} t1-t4 (90 80 2\*70);
- array test{4} \_TEMPORARY\_ (90 80 70 70);

### 例 3: 文字列の初期値を割り当てる

- array test2{\*} \$ a1 a2 a3 ('a','b','c');

### 例 4: 高度な配列を定義する

- array new{2:5} green jacobs denato fetzer;
- array x{5,3} score1-score15;
- array test{3:4,3:7} test1-test10;
- array temp{0:999} \_TEMPORARY\_;
- array x{10} (2\*1:5);

### 例 5: 先頭に 0 を持つ変数名の範囲を作成する

次の例では、先頭に 0 を持つ変数名の範囲を作成します。各変数名の長さは 3 文字です。この変数名は A01、A02、...A10 の順に正しく並べられます。先頭の 0 がなければ、変数名は、A1, A10, A2, ... A9 という順序で並べられます。

```
data test (drop=i);
  array a{10} A01-A10;
  do i=1 to 10;
    a{i}=i;
  end;
run;
proc print noobs data=test;
run;
```

### アウトプット 2.1 先頭に 0 を持つ配列名

A01	A02	A03	A04	A05	A06	A07	A08	A09	A10
1	2	3	4	5	6	7	8	9	10

### 関連項目:

#### ステートメント:

- “配列参照ステートメント”(25 ページ)

---

## 配列参照ステートメント

処理する配列要素を記述します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
情報

**種類:** 宣言

**制限事項:** VARCHAR データタイプの変数はサポートされていません。

### 構文

```
array-name {subscript};
```

### 引数

#### *array-name*

同一の DATA ステップ内で ARRAY ステートメントによってすでに定義されている配列名です。

#### {*subscript*}

*subscript*(添字)を指定します。次のいずれかの形式を使用できます。

#### {*variable-1*< , ...*variable-n*>}

通常、DO ループの処理で使用する変数または変数のリストを指定します。DO ループの実行時ごとに、この変数の現在の値が処理される配列要素の *subscript* になります。

**ヒント** *subscript* は中かっこ({})、大かっこ([ ])、丸かっこ(( ))で囲むことができます。

**例** “例 1: 反復 DO ループ処理を使用する”(27 ページ)

#### {\*}

配列内の要素を変数リストとして強制的に処理するように指定します。

**制限事項** 一時配列要素を含む配列を定義する際、アスタリスクを使用して配列要素を参照することはできません。

---

**ヒント** アスタリスクは、INPUT ステートメント、PUT ステートメント、および一部の SAS 関数で使用できます。

この構文は処理を容易にするために提供されていますが、通常の配列処理では使用できません。

**例** [“例 4: 変数リストとしてアスタリスクを使用する” \(27 ページ\)](#)

### *expression-1* < , ... *expression-n* >

SAS 式を指定します。

**範囲** 配列参照を含むステートメントの実行時、この式の値が subscript 値になります。また、この式の結果は、配列数の下限値と上限値の範囲内の整数になります。

**例** [“例 3: subscript を指定する” \(27 ページ\)](#)

## 詳細

- プログラムステートメント内で配列を参照するには、配列参照を使用します。配列を定義する ARRAY ステートメントは、DATA ステップ内でこの配列の参照箇所よりも前の位置に指定する必要があります。配列定義は同じ DATA ステップでのみ有効です。複数の DATA ステップで同じ配列を使用する場合、ステップごとに配列を再定義します。

### 注意:

- **SAS 関数の名前を配列名として使用すると、予期しない結果が生じることがあります。** 関数名を誤って配列名として使用すると、DATA ステップの実行時に、指定した名前を含むかっこ付きの参照名は関数の呼び出しではなく配列参照として解釈されます。この結果、警告メッセージが SAS ログに書き込まれます。
- 配列参照は、SAS 関数や SAS ステートメントなどの SAS 式を記述できる場所であればどこにでも指定できます。
  - 割り当てステートメント
  - 合計ステートメント
  - DO UNTIL(*expression*)
  - DO WHILE(*expression*)
  - IF
  - INPUT
  - PUT
  - SELECT
- 配列次元の下限値が 1 の場合、配列次元の要素数を返すために、DIM 関数を反復 DO ステートメントと組み合わせてよく使用します。DIM 関数を使用すると、DO ステートメントの上限値を変更せずに配列要素の数を変更することができます。次の例では、DIM(NEW)の結果として 4 が返されるため、後続のステートメントでは配列内のすべての要素を処理します。

```
array new{*} score1-score4;
do i=1 to dim(new);
  new{i}=new{i}+10;
end;
```

## 比較

ARRAY ステートメントでは配列を定義しますが、配列参照では処理対象となる配列メンバを定義します。

## 例

### 例 1: 反復 DO ループ処理を使用する

この例では、DO ループを繰り返す際、変数 I の値を配列参照の subscript として使用して、配列の各要素を処理します。配列要素の値が 99 の場合、その値は IF-THEN ステートメントによって 100 に変更されます。

```
array days{7} d1-d7;
do i=1 to 7;
  if days{i}=99 then days{i}=100;
end;
```

### 例 2: 1つのステートメントで複数の配列を参照する

1つの SAS ステートメントで複数の配列を参照することができます。この例では、DAYS と HOURS という 2つの配列を作成します。DO ループ内のステートメントでは、この 2つの配列にある配列要素を参照して変数 I の現在の値を置き換えます。

```
array days{7} d1-d7;
array hours{7} h1-h7;
do i=1 to 7;
  if days{i}=99 then days{i}=100;
  hours{i}=days{i}*24;
end;
```

### 例 3: subscript を指定する

この例では、INPUT ステートメントを使用して、変数 A1 と変数 A2、配列 ARR1 の 3番目の要素(A3)を読み取ります。

```
array arr1{*} a1-a3;
x=1;
input a1 a2 arr1{x+2};
```

### 例 4: 変数リストとしてアスタリスクを使用する

- array cost{10} cost1-cost10;  
totcost=sum(of cost {\*});
- array days{7} d1-d7;  
input days {\*};
- array hours{7} h1-h7;  
put hours {\*};

## 関連項目:

### 関数:

- [“DIM Function” \(SAS Viya Functions and CALL Routines: Reference\)](#)

### ステートメント:

- [“ARRAY ステートメント” \(20 ページ\)](#)
- [“DO ステートメント、反復” \(63 ページ\)](#)

---

## 割り当てステートメント

式を評価し、その結果を変数に格納します。

該当要素:	DATA ステップ
カテゴリ:	アクション CAS
種類:	実行

---

### 構文

*variable*=*expression*;

### 引数

#### *variable*

新しい変数名または既存の変数名を指定します。 .

範囲 *variable* には、変数名、配列参照、SUBSTR 関数を指定できます。

ヒント 割り当てステートメントで作成された変数は自動的に保持されません。

#### *expression*

SAS 式を指定します。

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

### 詳細

割り当てステートメントは、等号の右側の式を評価し、等号の左側に指定した変数に結果を格納します。

### 例: さまざまな式を使用した割り当てステートメント

さまざまな式を使用した割り当てステートメントを次に示します。

- name='Amanda Jones';
- WholeName='Ms. '||name;
- a=a+b;

### 関連項目:

#### ステートメント:

- [“合計ステートメント” \(394 ページ\)](#)



## ATTRIB ステートメント

1 つまたは複数の変数に出力形式、入力形式、ラベル、長さを設定します。

<b>該当要素:</b>	DATA ステップ
<b>カテゴリ:</b>	情報
<b>種類:</b>	宣言
<b>制限事項:</b>	このステートメントは CAS Server ではサポートされていません。

### 構文

**ATTRIB** *variable-list(s) attribute-list(s)* ;

### 引数

#### **variable-list(s)**

属性を設定する変数を指定します。

**ヒント** SAS で使用可能な任意の形式で変数をリストします。

#### **attribute-list(s)**

*variable-list* に割り当てる 1 つまたは複数の属性を指定します。ATTRIB ステートメントでは、次の属性を 1 つ以上指定します。

#### **FORMAT=format**

*variable-list* の変数に出力形式を割り当てます。

**ヒント** この出力形式は、標準の SAS 出力形式でも FORMAT プロシジャで定義した出力形式でもかまいません。

#### **INFORMAT=informat**

*variable-list* の変数に入力形式を割り当てます。

**ヒント** この入力形式は、標準の SAS 入力形式でも FORMAT プロシジャで定義した入力形式でもかまいません。

#### **LABEL='label'**

*variable-list* の変数にラベルを割り当てます。

#### **LENGTH=<\$>length**

*variable-list* の変数に長さを指定します。

**範囲** 文字変数の場合は、すべての動作環境で 1 バイトから 32,767 バイトまで指定できます。

**制限事項** DATASETS プロシジャから LENGTH= を使用して変数の長さを変更することはできません。

**要件** 文字変数の長さの前には、ドル記号(\$)を配置します。

**動作環境** 数値変数に指定できる長さの最小値は、ユーザーのシステムが使用する浮動小数点形式によって異なります。ほとんどのシステムでは IEEE 浮動小数点形式が使用され、最小値は 3 バイトです。

**ヒント** 既存のデータセットを入力値として使用する場合、出力データセットでの変数の長さを変更するには、SET ステートメントの前に ATTRIB ステートメントを使用します。

### TRANSCODE=YES | NO

文字変数をトランスコードするかどうかを指定します。トランスコードを実行しないようにするには、TRANSCODE=NO と指定します。

**デフォルト** YES

**制限事項** SAS Workspace Server クライアントによっては、TRANSCODE=NO の属性がサポートされていない場合があります。SAS 9.2 では、この属性がサポートされていない場合、TRANSCODE=NO と指定した変数の値はアスタリスク(\*)で置き換えられます(マスクされます)。SAS 9.2 より前のリリースでは、TRANSCODE=NO と指定した変数もトランスコードされていました。

旧リリースの SAS では、TRANSCODE=NO 属性を指定した変数が含まれる SAS 9.1 データセットにアクセスすることはできません。

V6TAPE Engine では、トランスコードの抑制はサポートされていません。

**操作** [“VTRANSCODE Function” \(SAS Viya National Language Support: Reference Guide\)](#) および [“VTRANSCODEX Function” \(SAS Viya National Language Support: Reference Guide\)](#) を使用すると、文字変数がトランスコードされるかどうかを示す値を返すことができます。

データセット内に TRANSCODE=属性が NO に設定されている文字変数がある場合、CONTENTS プロシジャを実行すると、データセット内の各変数の TRANSCODE=の値を含むトランスコード列を出力します。データセット内のすべての変数で TRANSCODE=がデフォルト値(YES)に設定されている場合、トランスコード列は出力されません。

**参照項目** [“Transcoding for NLS” \(SAS Viya National Language Support: Reference Guide\)](#)

## 詳細

### 基本

DATA ステップで ATTRIB ステートメントを使用すると、変数を含む SAS データセットのディスクリプタ情報が変更されるため、変数と属性は恒久的に関連付けられます。

PROC ステップで ATTRIB ステートメントを使用することはできますが、規則が異なります。

## ATTRIB ステートメントで INFORMAT=オプションを使用して入力形式を指定した場合の変数の処理

ATTRIB ステートメントで INFORMAT=オプションを使用して変数に割り当てた入力形式は、修飾リスト入力で使用される入力形式と同じように機能します。変数はリスト入力の走査機能を使用して読み込まれますが、入力形式が適用されません。修飾リスト入力では、SAS は次の処理を行います。

- 入力形式の w の値を使用して、外部ファイルの列位置や入力フィールド幅を設定しない
- 入力形式の w の値を使用して、事前に定義されていない文字変数の長さを設定する
- 数値入力形式の w の値は無視する
- 入力形式の d の値を、数値入力形式の場合と同じように使用する
- 入力データに埋め込まれている空白は、INFILE ステートメントの DLM=オプションまたは DLMSTR=オプションを変更しない限り、区切り文字として処理する

フォーマット入力やカラム入力などの別の入力スタイルを使用するように INPUT ステートメントをコード化した場合、ATTRIB ステートメントの INFORMAT オプションの使用時にはその入力スタイルは使用されません。

## SET ステートメントおよび MERGE ステートメントの使用時のトランスコード変数の処理

SET または MERGE ステートメントを使用して複数のデータセットからデータセットを 1 つ作成する場合、SAS では、出力データセット内の変数の TRANSCODE=属性が最初のデータセット内の変数の TRANSCODE=属性と同じ値に設定されます。“例 2: SET ステートメントをトランスコード変数と組み合わせて使用する” (32 ページ) および “例 3: MERGE ステートメントをトランスコード変数と組み合わせて使用する” (32 ページ) を参照してください。

注: TRANSCODE=属性は、変数が入力データセットまたは ATTRIB TRANSCODE=ステートメントで最初に検出されたときに設定されます。SET または MERGE ステートメントが ATTRIB TRANSCODE=属性の前に使用されている場合に、この TRANSCODE=属性が SET ステートメントと矛盾すると、警告メッセージが表示されます。

## 比較

変数に割り当てた属性を変更する場合、ATTRIB ステートメントを使用することも、または FORMAT、INFORMAT、LABEL、LENGTH などの個々の属性のステートメントを使用することもできます。

## 例

### 例 1: 指定する変数と属性の数が異なる ATTRIB ステートメントの例

指定する変数と属性の数が異なる ATTRIB ステートメントの例を次に示します。

- 1 変数と 1 属性:  
attrib cost length=4;
- 1 つの変数と複数の属性:  
attrib saleday informat=mmddy.  
format=worddate.;

- 複数の変数と同じ複数の属性:  
attrib x y length=\$4 label='TEST VARIABLE';
- 複数の変数と異なる複数の属性:  
attrib x length=\$4 label='TEST VARIABLE'  
y length=\$2 label='RESPONSE';
- 変数リストと 1 つの属性:  
attrib month1-month12  
label='MONTHLY SALES';

### 例 2: SET ステートメントをトランスコード変数と組み合わせて使用する

SET ステートメントを使用しているこの例では、最初のデータセットが B であり、データセット B の変数 Z の TRANSCODE=属性が NO に設定されています。そのため、データセット A の変数 Z の TRANSCODE=属性は NO になります。

```
data b;
  length z $4;
  z = 'ice';
  attrib z transcode = no;
data c;
  length z $4;
  z = 'snow';
  attrib z transcode = yes;
data a;
  set b;
  set c;
  /* Check transcode setting for variable Z */
  rc1 = vtranscode(z);
  put rc1=;
run;
```

### 例 3: MERGE ステートメントをトランスコード変数と組み合わせて使用する

MERGE ステートメントを使用しているこの例では、C が最初のデータセットで、このデータセット C の変数 Z の TRANSCODE=属性が YES に設定されています。そのため、データセット A の変数 Z の TRANSCODE=属性は NO になります。

```
data b;
  length z $4;
  z = 'ice';
  attrib z transcode = no;
data c;
  length z $4;
  z = 'snow';
  attrib z transcode = yes;
data a;
  merge c b;
  /* Check transcode setting for variable Z */
  rc1 = vtranscode(z);
  put rc1=;
run;
```

**関連項目:**

- [“Numeric Variable Length and Precision in Linux Environments” \(Batch and Line Mode Processing in SAS Viya\)](#)

**関数:**

- [“VTRANSCODE Function” \(SAS Viya National Language Support: Reference Guide\)](#)
- [“VTRANSCODEX Function” \(SAS Viya National Language Support: Reference Guide\)](#)

**ステートメント:**

- [“FORMAT ステートメント” \(129 ページ\)](#)
- [“INFORMAT ステートメント” \(176 ページ\)](#)
- [“LABEL ステートメント” \(217 ページ\)](#)
- [“LENGTH ステートメント” \(222 ページ\)](#)

---

**BY ステートメント**

DATA ステップ内の SET、MERGE、MODIFY、UPDATE の各ステートメントの実行を制御し、特別なグループ変数を設定します。

**該当要素:** DATA ステップまたは PROC ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 宣言

---

**構文**

```
BY <DESCENDING> variable-1 <...<DESCENDING> variable-n>
<NOTSORTED> <GROUPFORMAT>;
```

**引数****DESCENDING**

指定した変数の値を基にデータセットを降順で並べ替えるように指定します。降順では、数値変数であれば最大値から最小値に、文字変数であれば逆アルファベット順に並べ替えが実行されます。.

**制限事項** DESCENDING は CAS で実行中の DATA ステップではサポートされていません。

インデックス付きデータセットには DESCENDING オプションを使用できません。これは、インデックスが常に昇順で保存されるためです。

**例** [“例 2: 並べ替え順序の指定” \(37 ページ\)](#)

---

**GROUPFORMAT**

BY 変数の内部値ではなく、フォーマット値を使用して、BY グループの開始と終了が指定されます。このため、FIRST.変数と LAST.変数がどのように割り当てられるかが指定されます。GROUPFORMAT オプションは BY ステートメントのどの位置に指定してもかまいません。このオプションは、BY ステートメントにあるすべての変数に適用されます。

**制限事項** BY ステートメントで GROUPFORMAT オプションを使用する前に、BY 変数の値に基づいてデータセット内のオブザベーションを並べ替える必要があります。

BY ステートメントの GROUPFORMAT オプションは、DATA ステップのみで使用できます。

**操作** また、NOTSORTED オプションを使用する場合、データセットを並べ替えたり、インデックスを作成することなく、データセット内のオブザベーションを BY 変数のフォーマット値に基づいてグループ化することができます。

**注** GROUPFORMAT を使用した DATA ステップでの BY グループ処理は、SAS プロシジャでのフォーマット値の BY グループ処理と同じです。

**ヒント** GROUPFORMAT オプションは、グループ化するデータの表示に独自の出力形式を定義する場合に便利です。

DATA ステップで GROUPFORMAT オプションを使用すると、データセットの作成に使用する BY グループと、グループ化され、フォーマットされたデータをレポートする PROC ステップの BY グループを一致させることができます。

**例** “例 4: フォーマット値を用いてオブザベーションをグループ化する” (37 ページ)

**variable**

データセットの並べ替えやインデックス付けに使用される変数を指定します。指定した変数は、現在の DATA ステップまたは PROC ステップの BY 変数として参照されます。

**要件** BY グループ処理で BY 変数として名前リテラルを指定し、対応する FIRST.一時変数または LAST.一時変数を参照する場合、FIRST.または LAST.が含まれる 2 レベル変数名を一重引用符で囲んで指定する必要があります。次に、例を示します。

```
data sedanTypes;
  set cars;
  by 'Sedan Types'n;
  if 'first.Sedan Types'n then type=1;
run;
```

**ヒント** データセットでは、複数の変数を基準に並べ替えやインデックスの作成を実行できます。

**例** “例 1: BY 変数の指定” (36 ページ)

“例 2: 並べ替え順序の指定” (37 ページ)

“例 3: 並べ替えを実行していないデータの BY グループ処理” (37 ページ)

“例 4: フォーマット値を用いてオブザベーションをグループ化する” (37 ページ)

## NOTSORTED

同じ BY 変数の値を含むオブザベーションをグループ化します。ただし、これらのオブザベーションはアルファベット順または数値順では並べ替えられません。

**制限事項** MERGE および UPDATE ステートメントでは、NOTSORTED オプションを使用できません。

**ヒント** NOTSORTED オプションは BY ステートメントのどの位置に指定してもかまいません。

NOTSORTED オプションは、日付順やカテゴリなど他の論理グループに分類されるデータが存在する場合に便利です。

**例** “例 3: 並べ替えを実行していないデータの BY グループ処理” (37 ページ)

## 詳細

### BY グループの開始と終了の識別について

BY グループの開始と終了は、BY 変数に 2 つの一時変数を作成して識別します。FIRST.variable and LAST.variable. この 2 つの変数の値は 0 または 1 のどちらかになります。FIRST.変数の値は、BY グループの最初のオブザベーションを読み込むときに 1 に設定されます。LAST.変数の値は、BY グループの最後のオブザベーションを読み込むときに 1 に設定されます。これら 2 つの一時変数は、DATA ステップでのプログラミングには利用できますが、出力データセットには変数として書き出されません。

### DATA ステップ

BY ステートメントは、DATA ステップでその前に指定されている SET、MERGE、MODIFY、UPDATE のステートメントに適用されます。ただし、DATA ステップ内のこれらの各ステートメントに指定できる BY ステートメントは 1 つのみです。

SET、MERGE、UPDATE のステートメントにリストされたデータセットは、BY ステートメントにリストされている変数の値で並べ替えられているか、適切なインデックスを持つ必要があります。デフォルトでは、データセットは昇順の数値順またはアルファベット順で並べられていると判断されます。オブザベーションは次のいずれかの方法で配置されます。

- データセットを並べ替える
- 変数のインデックスを作成する
- オブザベーションを順番に読み込む

**注:** MODIFY ステートメントではデータを並べ替える必要はありませんが、並べ替えを実行するとパフォーマンスを向上させることができます。

**注:** BY ステートメントでは、SORTSEQ=LINGUISTIC オプションが指定された SORT プロシジャを使用して並べ替えたデータの言語照合が優先されます。

### PROC ステップ

BY ステートメントを一部の SAS プロシジャで使用すると、その SAS プロシジャのアクションを変更することができます。BY ステートメントが SAS プロシジャの処理に与える影響については、[SAS Viya Data Management and Utility Procedures Guide](#) にある各プロシジャの説明を参照してください。

### SAS ビュー

DBMS からの読み込みに基づき DATA ステップビューを作成する際、SET、MERGE、UPDATE、MODIFY の各ステートメントの後ろに BY ステートメントが指定されている場合、BY ステートメントでは DBMS 側でデータを並べ替えを行い、並べ替えたデータを返すことがあります。データの並べ替えが発生すると、実行時間が長くなります。

### BY グループの処理

FIRST.変数と LAST.変数には、次の値が割り当てられます。

- FIRST.変数は、次の場合に値が 1 に設定されます。
  - 現在のオブザベーションがデータセットから読み込む最初のオブザベーションである場合。
  - GROUPFORMAT オプションを使用していないときに、現在のオブザベーションにある変数の内部値が前のオブザベーションにある内部値と異なる場合。

GROUPFORMAT オプションを使用しているとき、FIRST.変数が 1 に設定されるのは、現在のオブザベーションにある変数のフォーマット値が前のオブザベーションにあるフォーマット値と異なる場合です。

- FIRST.変数が、BY ステートメント内でその前にあるどの変数に対しても 1 に設定される場合。

該当しない場合、FIRST.変数の値は 0 になります。

- LAST.変数は、次の場合に値が 1 に設定されます。
  - 現在のオブザベーションがデータセットから読み込む最後のオブザベーションである場合。
  - GROUPFORMAT オプションが使用されているときに、現在のオブザベーションにある変数の内部値が次のオブザベーションにある内部値と異なる場合。

GROUPFORMAT オプションが使用されている場合、LAST.変数が 1 に設定されるのは、現在のオブザベーションにある変数のフォーマット値が次のオブザベーションにあるフォーマット値と異なる場合です。

- LAST.変数が、BY ステートメント内でその前にあるどの変数に対しても 1 に設定される場合。

該当しない場合、LAST.変数の値は 0 になります。

## 例

### 例 1: BY 変数の指定

- オブザベーションは DEPT 変数の昇順で並べられます。

```
by dept;
```



- オブザベーションは CITY のアルファベット順(昇順)で並べられます。CITY の値内では、ZIPCODE の昇順で並べられます。

```
by city zipcode;
```

### 例 2: 並べ替え順序の指定

注: DESCENDING は CAS で実行中の DATA ステップではサポートされていません。

- オブザベーションは SALESREP の昇順で並べられます。SALESREP の値内では、JANSALES の値の降順で並べられます。

```
by salesrep descending jansales;
```

- オブザベーションは BEDROOMS の降順で並べられます。BEDROOMS の値内では、PRICE の値の降順で並べられます。

```
by descending bedrooms descending price;
```

### 例 3: 並べ替えを実行していないデータの BY グループ処理

オブザベーションは費用が発生した月の名前で並べられます。

```
by month notsorted;
```

### 例 4: フォーマット値を用いてオブザベーションをグループ化する

次に GROUPFORMAT オプションを使用例を示します。

```
proc format;
  value range
    low -55 = 'Under 55'
    55-60 = '55 to 60'
    60-65 = '60 to 65'
    65-70 = '65 to 70'
    other = 'Over 70';
run;
proc sort data=sashelp.class out=sorted_class;
  by height;
run;
data _null_;
  format height range.;
  set sorted_class;
  by height groupformat;
  if first.height then
    put 'Shortest in ' height 'measures ' height:best12.;
run;
```

SAS は次の出力をログに書き込みます。

```
Shortest in Under 55 measures 51.3 Shortest in 55 to 60 measures 56.3 Shortest in 60 to 65 measures
62.5 Shortest in 65 to 70 measures 65.3 Shortest in Over 70 measures 72
```

### 例 5: 複数のオブザベーションを結合した後で 1 つの BY の値を基にグループ化する

次の例では、BY グループの処理で FIRST.変数と LAST.変数を使用する方法を示します。

```
data Inventory;
  length RecordID 8 Invoice $ 30 ItemLine $ 50;
```

```

infile datalines;
input RecordID Invoice ItemLine &;
drop RecordID;
datalines;
A74 A5296 Highlighters
A75 A5296 Lot # 7603
A76 A5296 Yellow Blue Green
A77 A5296 24 per box
A78 A5297 Paper Clips
A79 A5297 Lot # 7423
A80 A5297 Small Medium Large
A81 A5298 Gluestick
A82 A5298 Lot # 4422
A83 A5298 New item
A84 A5299 Rubber bands
A85 A5299 Lot # 7892
A86 A5299 Wide width, Narrow width
A87 A5299 1000 per box
;
data combined;
array Line{4} $ 60 ;
retain Line1-Line4;
keep Invoice Line1-Line4;
set Inventory;
by Invoice;

if first.Invoice then do;
  call missing(of Line1-Line4);
  records = 0;
end;

records + 1;
Line[records]=ItemLine;

if last.Invoice then output;
run;
proc print data=combined;
title 'Office Supply Inventory';
run;

```

**アウトプット 2.2** 結合した複数のオブザベーションからの出力

Office Supply Inventory					
Obs	Line1	Line2	Line3	Line4	Invoice
1	Highlighters	Lot # 7603	Yellow Blue Green	24 per box	A5296
2	Paper Clips	Lot # 7423	Small Medium Large		A5297
3	Gluestick	Lot # 4422	New item		A5298
4	Rubber bands	Lot # 7892	Wide width, Narrow width	1000 per box	A5299

**関連項目:****ステートメント:**

- [“MERGE ステートメント” \(280 ページ\)](#)
- [“MODIFY ステートメント” \(289 ページ\)](#)
- [“SET ステートメント” \(378 ページ\)](#)
- [“UPDATE ステートメント” \(408 ページ\)](#)

---

**CALL ステートメント**

SAS CALL ルーチンを呼び出します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション  
CAS

**種類:** 実行

---

**構文**

**CALL** *routine*(*parameter-1* <, ...*parameter-n*>);

**引数*****routine***

呼び出したい SAS CALL ルーチンの名前を指定します。

**参照項目:** 使用可能なルーチンの詳細については、[SAS Viya 関数と CALL ルーチン: リファレンス](#)を参照してください。

---

**(*parameter*)**

ルーチンに渡す情報、またはルーチンから返される情報を指定します。

**要件** ルーチンによっては、この情報を丸かっこで囲む必要があります。

---

**ヒント** カンマで区切ることにより、他のパラメータを追加指定することができます。

---

**詳細**

SAS CALL ルーチンを使用すると、値を割り当てたり、その他のシステム機能を実行できます。

**関連項目:**

[SAS Viya 関数と CALL ルーチン: リファレンス](#)

---

**CARDS ステートメント**

データ行の始まりを示します。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	宣言
別名:	DATALINES, LINES
制限事項:	このステートメントは CAS Server ではサポートされていません。
参照項目:	<a href="#">“DATALINES ステートメント” (57 ページ)</a>

---

## CARDS4 ステートメント

セミコロンが含まれるデータ行の始まりを示します。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	宣言
別名:	DATALINES4, LINES4
制限事項:	このステートメントは CAS Server ではサポートされていません。
参照項目:	<a href="#">“DATALINES4 ステートメント” (59 ページ)</a>

---

## CATNAME ステートメント

複数のカタログをカタログ参照名(ショートカット名)に関連付け、1つのカタログに論理的に結合します。また、1つまたはすべてのカタログ参照名をクリアします。1つの連結またはすべての連結に含まれる連結カタログをリストします。

該当要素:	任意の場所
カテゴリ:	データアクセス
制限事項:	このステートメントは CAS Server ではサポートされていません。

### 構文

```
CATNAME <libref.> catref
    <(libref-1.catalog-1 <(ACCESS=READONLY)>
    <...libref-n.catalog-n <(ACCESS=READONLY)>>>;
CATNAME <libref.> catref CLEAR | _ALL_ CLEAR;
CATNAME <libref.> catref LIST | _ALL_ LIST;
```

### 引数

#### *libref*

割り当て済みの SAS ライブラリ参照名です。ライブラリ参照名を指定しない場合、WORK ライブラリにあるカタログを、指定のカタログ参照名を使用して連結します。

制限事項 ライブラリ参照名は事前に割り当てる必要があります。

---

**catref**

ステートメントに指定するカタログまたはカタログ連結に対する、重複しないカタログ参照名です。*libref.catref*のように、ライブラリ参照名とカタログ参照名はピリオドで区切ります。このカタログ参照名には、任意の SAS 名を使用できます。

**catalog**

カタログ連結に使用できるカタログ名です。

**オプション****CLEAR**

現在割り当てられている *catref* または *libref.catref* の関連付けを取り消します。

**ヒン** 特定の *catref* または *libref.catref* を指定すると、1 つの連結からその関連付けを取り消します。**\_ALL\_ CLEAR** を指定すると、現在割り当てられているすべての *catref* または *libref.catref* の関連付けを取り消します。

**\_ALL\_ CLEAR**

現在割り当てられているすべての *catref* または *libref.catref* の連結を取り消します。

**LIST**

指定した連結に含まれるカタログ名を SAS ログに書き込みます。

**ヒン** *catref* または *libref.catref* を指定すると、1 つの連結の属性を書き込みます。**\_ALL\_** を指定すると、現在のセッションにあるすべてのカタログ連結の属性を書き込みます。

**\_ALL\_ LIST**

現在のカタログの連結に含まれているすべてのカタログ名を SAS ログに書き込みます。

**ACCESS=READONLY**

カタログに読み取り専用属性を割り当てます。この属性を割り当てると、ユーザーはカタログエントリを読み取ることはできますが、情報の更新や新しい情報の書き込みは行えなくなります。

**詳細****CATNAME を使用する理由**

CATNAME を使用すると、1 つのカタログ参照名(*libref.catref* または *catref*)を指定することによって、複数のカタログ内のエントリにアクセスできるので便利です。カタログ連結を作成すると、単純な(連結されていない)カタログ参照名を使用できるコンテキストであればカタログ参照名を指定できます。

**カタログ連結のルール**

カタログ連結を効果的に使用するには、連結カタログ間でのカタログエントリの検索ルールを理解する必要があります。

- 入力や更新を行うためにカタログエントリを開く場合、連結カタログが検索され、最初に検索された指定エントリが使用されます。
- 出力するためにカタログエントリを開く場合、カタログエントリは連結に最初にリストされたカタログ内に作成されます。

注 連結の他のカタログに同じ名前のエントリが存在する場合でも、新しいエントリは最初のカタログの中に作成されます。

注 連結の最初のカタログを更新するために開く際にこのカタログが存在しない場合、存在する連結の次のカタログに書き込まれます。

- カタログエントリの削除や名前の変更を行う場合、最初に検出されたエントリにのみ影響します。
- カタログエントリのリストを表示する場合、カタログエントリ名は常に1つだけ表示されます。

注 連結内に同じ名前が複数存在する場合でも、最初に検索されたエントリのみが表示されます。

## 比較

- CATNAME ステートメントは、カタログ向けの LIBNAME ステートメントのようなものです。LIBNAME ステートメントを使用すると SAS ライブラリにショートカット名を割り当てることができるため、ショートカット名を使用して、ファイルを検索して、ファイル内のデータを使用できます。CATNAME を使用すると、1つまたは複数のカタログに短縮名<libref.>catref (libref は任意)を割り当てることができるので、SAS ではこれらのカタログを検索して、各カタログのすべてまたは一部のエントリを使用できます。
- CATNAME ステートメントでは、*明示的に* SAS カタログが連結されます。LIBNAME ステートメントでは、*暗示的に* SAS カタログが連結されます。

## 例

### 例 1: カタログ連結の割り当てと使用

複数の SAS カタログ内のエントリにアクセスしなければならない場合もあります。情報にアクセスする最も効率的な方法は、カタログを論理的に連結することです。カタログを連結すると、新たに、サイズの大きいカタログを別に作成することなく情報にアクセスできます。

連結対象のカタログを含む SAS ライブラリに対してライブラリ参照名を割り当てるには、次のようにします。

```
libname mylib1 'data-library-1';
libname mylib2 'data-library-2';
```

論理的に連結するカタログのリストに対してカタログ参照名を割り当てます。この名前には有効な SAS 名を指定します。

```
catname allcats (mylib1.catalog1 mylib2.catalog2);
```

SAS ログには次のメッセージが表示されます。

#### ログ 2.1 CATNAME ステートメント実行後のログ出力

NOTE: Catalog concatenation WORK.ALLCATS has been created.

ライブラリ参照名が指定されていないため、ライブラリ参照名はデフォルトで WORK に設定されます。これらのカタログのいずれかにあるカタログエントリにアクセスする場合、元のライブラリ参照名とカタログ名ではなく、ライブラリ参照名 WORK とカタログ参照名 ALLCATS を使用します。たとえば、カタログ

MYLIB1.CATALOG1 にある APPKEYS.KEYS というカタログエントリにアクセスするには、次のように指定します。

```
work.allcats.appkeys.keys
```

## 例 2: ネストされたカタログ連結の作成

連結カタログを作成した後、CATNAME を使用して他の 1 つのカタログ、または他の連結されたカタログと現在の連結を結合することができます。ネストされたカタログ連結では、1 つのカタログ参照名を使用して複数の異なるカタログの組み合わせにアクセスできるので便利です。

```
libname local 'my_dir';
libname main 'public_dir';
catname private_catalog (local.my_application_code
                        local.my_frames
                        local.my_formats);
catname combined_catalogs (private_catalog
                           main.public_catalog);
```

前述の例では、PRIVATE\_CATALOG を使用すると、個人用のアプリケーションエントリのコピーで作業することができます。パブリック版のアプリケーションと連結した場合にエントリがどのように動作するか確認する場合は、COMBINED\_CATALOGS を使用できます。

## 関連項目:

### ステートメント:

- [“FILENAME ステートメント” \(96 ページ\)](#)
- [“LIBNAME ステートメント” \(225 ページ\)](#)(SAS カタログの暗示的な連結の説明)

---

## CHECKPOINT EXECUTE\_ALWAYS ステートメント

チェックポイント-再開データを無視して、このステートメントの直後にある DATA ステップまたは PROC ステップを実行するように指示します。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

## 構文

```
CHECKPOINT EXECUTE_ALWAYS;
```

### 引数なし

CHECKPOINT EXECUTE\_ALWAYS ステートメントは、チェックポイントデータを無視して、このステートメントの直後にある DATA ステップや PROC ステップを実行するように指示します。

## 詳細

チェックポイント-再開モードの適用時にバッチプログラムが完了前に強制終了した場合、強制終了が発生した時点で実行していた DATA ステップまたは PROC ステップからプログラムが再実行されます。バッチプログラムが強制終了する前に完了していた DATA ステップや PROC ステップは再実行されません。特定の DATA ステップや PROC ステップを再実行する必要がある場合、そのステップの前の位置に CHECKPOINT EXECUTE\_ALWAYS ステートメントを追加することができます。CHECKPOINT EXECUTE\_ALWAYS ステートメントを使用すると、チェックポイント-再開データを無視し、このステートメントに続くステップを実行します。

## 関連項目:

### システムオプション:

- ["STEPCHKPT System Option" \(SAS Viya System Options: Reference\)](#)
- ["STEPCHKPTLIB= System Option" \(SAS Viya System Options: Reference\)](#)
- ["STEPRESTART System Option" \(SAS Viya System Options: Reference\)](#)

---

## コメントステートメント

ステートメントまたはプログラムの目的を説明します。

**該当要素:** 任意の場所

**カテゴリ:** CAS  
ログ制御

## 構文

```
*message;
or
/*message*/
```

## 引数

***\*message;***

ステートメントまたはプログラムの説明や注釈となるテキストを指定します。

**範囲** このコメントの長さに制限はありません。セミコロンで終了します。

**制限事項** このコメントは独立したステートメントとして書き込む必要があります。

このコメント内にセミコロンは使用できません。

この形式のコメントに含まれるマクロステートメントやマクロ変数の参照は、SAS マクロ機能で処理されます。この形式のコメントでは、SAS マクロ機能によるテキストの処理を回避できません。

---



ヒント マクロ定義内にコメントを使用する場合や SAS マクロ機能によるテキストの処理を回避する場合は、次のスタイルのコメントを使用します。

```
/* message */
```

### ***/\*message\*/***

ステートメントまたはプログラムの説明や注釈となるテキストを指定します。

範囲 このコメントの長さに制限はありません。

制限事項 このタイプのコメントをネストすることはできません。

ヒント このコメントには、セミコロンや開始と終了が一致しない引用符を使用することができます。

このコメントは、ステートメント内や、SAS コード内で 1 つのブランクがある任意の場所に追加することができます。

## 詳細

SAS プログラム内の任意の場所にコメントステートメントを使用し、プログラムへの注釈、プログラムのわかりにくいセグメントに対する説明、複雑なプログラムや計算内のステップの説明を追加できます。処理実行中、コメントステートメントに指定されたテキストは無視されます。

### 注意:

コメントの開始を示す */\** は、列 1 と列 2 に指定しないようにしてください。動作環境によっては、列 1 および列 2 に指定されている */\** 記号を SAS プログラムまたはセッションの終了として扱うものがあります。

注: コードに次の行を追加すると、開始と終了が一致しないコメントタグや引用符、セミコロンの指定もれを修正することができます。

```
/* ; * "; */;
quit;
run;
```

## 例: コメントステートメントの使用

コメントには、次の 2 つのタイプがあります。

- この例では、*\*message;* の形式を使用します。

```
*This code finds the number in the BY group;
```

- この例では、*\*message;* の形式を使用します。

```
*-----*
| This uses one comment statement |
| to draw a box. |
*-----*;
```

- この例では、*/\*message\*/* の形式を使用します。

```
input @1 name $20. /* last name */
      @200 test 8. /* score test */
      @50 age 3.; /* customer age */
```

- この例では、*/\*message\*/* の形式を使用します。

```
/* For example 1 use: x=abc;  
   for example 2 use: y=ghi; */
```

---

## CONTINUE ステートメント

DO ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。

**該当要素:** DATA ステップ  
**カテゴリ:** CAS  
制御  
**種類:** 実行  
**制限事項:** DO ループでのみ使用できます。

---

### 構文

**CONTINUE;**

### 引数なし

CONTINUE ステートメントに引数はありません。このステートメントは、DO ループでのステートメントの現在の繰り返し処理を任意の条件に基づいて中止します。DO ループの次の繰り返しから処理を再開します。

### 比較

- CONTINUE ステートメントは、ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。LEAVE ステートメントは、現在のループ処理を終了します。
- CONTINUE ステートメントは DO ループのみで使用できます。LEAVE ステートメントは DO ループまたは SELECT グループで使用できます。

### 例: 他のステートメントの実行を回避する

この DATA ステップでは、正社員の福利厚生レポートを作成します。従業員のステータスが PT(パートタイム)の場合、この CONTINUE ステートメントでは 2 番目の INPUT ステートメントと OUTPUT ステートメントを実行しないようにします。

```
data new_emp;  
  drop i;  
  do i=1 to 5;  
    input name $ idno status $;  
    /* return to top of loop */  
    /* when condition is true */  
    if status='PT' then continue;  
    input benefits $10.;;  
    output;  
  end;  
  datalines;  
Jones 9011 PT  
Thomas 876 PT  
Richards 1002 FT
```

```

Eye/Dental
Kelly 85111 PT
Smith 433 FT
HMO
;

```

## 関連項目:

### ステートメント:

- [“DO ステートメント、反復” \(63 ページ\)](#)
- [“LEAVE ステートメント” \(221 ページ\)](#)

---

## DATA ステートメント

DATA ステップを開始します。また、出力データセット、ビュー、プログラムの名前を指定します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 宣言

---

## 構文

- 形式 1: **DATA** *<data-set-name-1 <(data-set-options-1)> >*  
*<...data-set-name-n <(data-set-options-n)> >*  
*</ <DEBUG> <NESTING> <STACK = stack-size> > <NOLIST>;*
- 形式 2: **DATA** *\_NULL\_ </ <DEBUG> <NESTING> <STACK = stack-size> > <NOLIST>;*
- 形式 3: **DATA** *view-name <data-set-name-1 <(data-set-options-1)> >*  
*<...data-set-name-n <(data-set-options-n)> > /*  
*VIEW=view-name <(password-option) <SOURCE=source-option> >*  
*<NESTING> <NOLIST>;*
- 形式 4: **DATA** *data-set-name / PGM=program-name <(password-option)*  
*<SOURCE=source-option> > <NESTING> <NOLIST>;*
- 形式 5: **DATA** *VIEW=view-name <(password-option)> <NOLIST>;*  
**DESCRIBE;**
- 形式 6: **DATA** *PGM=program-name <(password-option)> <NOLIST>;*  
*<DESCRIBE;>*  
*<REDIRECT INPUT | OUTPUT old-name-1 = new-name-1*  
*<... old-name-n = new-name-n> ;>*  
*<EXECUTE;>*
- 形式 7: **DATA** *<data-set-name-1 <(data-set-options-1)> >*  
*<...data-set-name-n <(data-set-options-n)> >*  
*/< <SESSREF=cas-session-reference-name> | <SESSUUID=cas-session-uuid> >*  
*<THREADS=number-of-threads> <SINGLE= NO | YES | NOINPUT>*  
*<NESTING> <STACK = stack-size><NOLIST>;*
- 形式 8: **DATA** *\_NULL\_*

```

/< <SESSREF=cas-session-reference-name> | <SESSUUID=cas-session-uuid> >
<THREADS=number-of-threads> <SINGLE= NO | YES | NOINPUT>
<NESTING> <STACK = stack-size> <NOLIST>;

```

### 引数なし

引数の指定を省略すると、作成される一連のデータセットには、DATA $n$  という名前が自動的に指定されます。 $n$  には、名前が重複しないように最小の整数が使用されます。

### 引数

#### **data-set-name**

DATA ステップで作成する SAS データファイルまたは DATA ステップビューの名前を指定します。DATA ステップビューを作成するには、*data-set-name* を少なくとも 1 つ指定する必要があります。また、*data-set-name* は *view-name* に一致させる必要があります。

**制限事項** *data-set-name* は SAS の命名規則に準拠する必要があります。また、動作環境によっては、その他の制限事項が適用される場合があります。

**ヒント** データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

SAS データセットを作成せずに、DATA ステップを実行することもできます。“例 5: カスタムレポートの作成” (55 ページ) を参照してください。詳細については、“データセットを作成しない場合(形式 2)” (53 ページ) を参照してください。

#### **(data-set-options)**

DATA ステップでオブザベーションを出力データセットに書き込むときに適用する引数を指定します。この引数はオプションです。

**参照項目:** [SAS Viya Data Set Options: Reference](#) には、データセットオプションの定義や一覧が記載されています。

**例** “例 1: 複数データファイルの作成とデータセットオプションの使用” (54 ページ)

#### **/ DEBUG**

論理エラーやデータエラーを特定しながら、対話的にプログラムのデバッグを行うことができます。

**制限事項** この引数は CAS Server ではサポートされていません。

#### **/ NESTING**

DO-END および SELECT-END の各ネストレベルの開始と終了に関するメッセージを SAS ログに出力できます。このオプションを指定すると、一致していない DO-END ステートメントおよび SELECT-END ステートメントをデバッグすることができます。また、ネストレベルがわかりにくい大規模なプログラムに使用すると便利です。

#### **/ STACK=stack-size**

ネストする LINK ステートメントの最大数を指定します。

**NULL**

DATA ステップの実行時にデータセットを作成しないように指定します。

**VIEW=view-name**

DATA ステップで入力 DATA ステップビューの格納に使用するビューの名前を指定します。

制限事項 この引数は CAS Server ではサポートされていません。

制限事項

---

VARCHAR データタイプの変数はサポートされていません。

---

*view-name* は、複数のデータセット名のうちの 1 つと一致している必要があります。

---

1 つの DATA ステップで作成できるビューは 1 つだけです。

ヒント DATA ステートメントに追加データセットを指定する場合、後続の DATA ステップまたは PROC ステップでビューが処理されるときに追加データセットが作成されます。ビューには、ビューの実行時に他のデータセットを生成する機能があります。

---

ビューの作成時、SAS マクロ変数が解決されます。マクロ変数の解決をビューの処理時まで延期する場合は、SYMGET 関数を使用します。

例 “例 2: 入力 DATA ステップビューの作成” (54 ページ)

---

“例 3: ビューおよびデータファイルの作成” (55 ページ)

**password-option**

コンパイル済み DATA ステッププログラムまたは DATA ステップビューにパスワードを割り当てます。

注: パスワードで保護された DATA ステッププログラムを表示するには、パスワードを指定する必要があります。プログラムが複数のパスワードを持っている場合、一番厳格なパスワードを指定します。ALTER は一番厳格で、READ は最も制限が緩いものです。詳細については、“[DESCRIBE ステートメント](#)” (61 ページ) を参照してください。

次のパスワードオプションを指定できます。

**ALTER=alter-password**

SAS データファイルに ALTER パスワードを割り当てます。このパスワードを使用すると、コンパイル済み DATA ステッププログラムまたは DATA ステップビューの保護や置き換えが行えます。

別名 PROTECT=

要件 コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に ALTER パスワードを使用する場合、プログラムやビューの置き換えには ALTER パスワードが必要です。

---

コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に ALTER パスワードを使用する場合、DESCRIBE ステートメントを実行するには ALTER パスワードが必要です。

**READ=***read-password*

SAS データファイルに READ パスワードを割り当てます。このパスワードを使用すると、コンパイル済み DATA ステッププログラムまたは DATA ステップビューの読み取りや実行を行えます。

別名 EXECUTE=

要件 コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に READ パスワードを使用した場合、そのプログラムやビューを実行するには READ パスワードが必要となります。

コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に READ パスワードを使用する場合、DESCRIBE ステートメントおよび EXECUTE ステートメントを実行するには READ パスワードが必要です。無効なパスワードを指定すると、DESCRIBE ステートメントだけが実行されます。

ヒント コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に READ パスワードを使用する場合、プログラムやビューの置き換えにはパスワードは不要です。

**PW=***password*

READ パスワードと ALTER パスワードを割り当てます。両方のパスワードに同じ値を使用します。

**SOURCE=***source-option*

次のソースオプションから 1 つ指定します。

**SAVE**

コンパイル済み DATA ステッププログラムや DATA ステップビューを作成したソースコードを保存します。

**ENCRYPT**

コンパイル済み DATA ステッププログラムや DATA ステップビューを作成したソースコードを暗号化して保存します。

ヒント ソースコードを暗号化する場合は、ALTER パスワードオプションも使用する必要があります。ALTER パスワードを使用しないと、警告メッセージが表示されます。

**NOSAVE**

ソースコードは保存されません。

## 注意:

DATA ステップビューに NOSAVE オプションを使用する場合、このビューを SAS のあるバージョンから別のバージョンにコピーすることはできません。

デフォルト SAVE

**PGM=***program-name*

DATA ステップで作成または実行するコンパイル済みプログラムの名前を指定します。コンパイル済みプログラムを作成する場合は、PGM=オプションの前にスラッシュ(/)を指定します。コンパイル済みプログラムを実行する場合は、スラッシュ(/)をつけずに PGM=オプションを指定します。

制限事項 この引数は CAS Server ではサポートされていません。

**ヒント** ストアドプログラムの作成時、SAS マクロ変数の解決が行われます。マクロ変数の解決をビューの処理時まで延期する場合は、SYMGET 関数を使用します。

**例** “例 4: コンパイル済みプログラムの保存と実行” (55 ページ)

### **SESSREF=cas-session-reference-name**

DATA ステップを CAS セッションに関連付けます。SESSREF=オプションの値は、現在のアクティブなセッションを表します。

**操作** DSACCEL システムオプションが NONE に設定されている場合、CAS で DATA ステップを実行するためには、SESSREF=または SESSUUID=オプションのどちらかが必要となります。

### **SESSUUID=cas-session-uuid**

DATA ステップが実行されている CAS セッションを指定します。cas-session-uuid は、CAS セッションに関連付けられている UUID(Universally Unique Identifier)になります。セッションを作成し、そのセッション用の UUID を取得するには、CAS ステートメントまたは LIBNAME ステートメントを使用します。

**操作** DSACCEL システムオプションが NONE に設定されている場合、CAS で DATA ステップを実行するためには、SESSREF=または SESSUUID=オプションのどちらかが必要となります。

**ヒント** セッションを作成し、そのセッション用の UUID を取得するには、CAS ステートメントまたは LIBNAME ステートメントを使用します。

### **SINGLE=NO | YES | NOINPUT**

単一スレッドで DATA ステップを実行するかどうかを指定します。

#### **NO**

全スレッドで DATA ステップを実行します。4 つのワーカーが存在し、各ワーカーが 32 個のスレッドをサポートする場合、DATA ステップは 128 個のスレッドで実行されます。これがデフォルトの設定です。

#### **YES**

単一スレッドで DATA ステップを実行します。4 つのワーカーが存在し、各ワーカーが 32 個のスレッドをサポートする場合、DATA ステップは 1 個のスレッドで実行されます。テーブル I/O が発生すると、すべての行がこの単一スレッドに移動されます。大量のデータを単一スレッドで処理することは推奨されません。

#### **NOINPUT**

入力テーブルが存在しない場合にのみ、データステッププログラムを単一スレッドで実行します。このモードは、テーブルに対する出力データのみを生成する DATA ステップを記述する場合に便利です。

### **THREADS=number-of-threads**

CAS で DATA ステップを実行するために使用するスレッドの数を指定します。

**要件** この数字は 0 以上である必要があります。値ゼロは、許可されている最大のスレッド数を使用して実行することを意味します。

### **NOLIST**

ERROR\_の値が 1 の場合、すべての変数を SAS ログに出力しないようにします。

制限事項 NOLIST は DATA ステートメントの最後のオプションとして指定する必要があります。

## 詳細

### DATA ステートメントの使用

DATA ステートメントを使用すると、DATA ステップが開始されます。DATA ステートメントを使用すると、SAS データセット、データビュー、ストアドプログラムのような、さまざまなタイプの出力を作成できます。DATA ステートメントには、複数の出力を指定できます。ただし、データビューは1つしか指定できません。VIEW=オプション (49 ページ) を指定すると、ビューを作成できます。また、PGM=オプション (50 ページ) を指定すると、ストアドプログラムを作成できます。

### READ パスワードと ALTER パスワードの両方を使用する

コンパイル済み DATA ステッププログラムまたは DATA ステップビューの作成時に READ パスワードと ALTER パスワードを両方使用すると、次のようになります。

- コンパイル済み DATA ステッププログラムまたは DATA ステップビューを実行するには、READ パスワードまたは ALTER パスワードが必要になります。
- コンパイル済み DATA ステッププログラムまたは DATA ステップビューに DESCRIBE ステートメントおよび EXECUTE ステートメントが含まれている場合は、READ パスワードまたは ALTER パスワードが必要になります。
- ALTER パスワードをこの DESCRIBE ステートメントおよび EXECUTE ステートメントに使用する場合、次のようになります。
  - DESCRIBE ステートメントと EXECUTE ステートメントの両方が実行されます。
  - 無効な ALTER パスワードを指定してコンパイル済み DATA ステッププログラムや DATA ステップビューを実行した場合、次のようになります。

DESCRIBE ステートメントは実行されません。

バッチモードでは、この EXECUTE ステートメントは何も影響しません。

対話型モードでは、READ パスワードの入力を求めるプロンプトが表示されます。READ パスワードが有効な場合、EXECUTE ステートメントが処理されます。READ パスワードが無効な場合、EXECUTE ステートメントは処理されません。

- READ パスワードを DESCRIBE ステートメントおよび EXECUTE ステートメントに使用すると、次のようになります。
  - 対話型モードでは、ALTER パスワードの入力を求めるプロンプトが表示されます。

有効な ALTER パスワードを入力すると、DESCRIBE ステートメントと EXECUTE ステートメントの両方が実行されます。

無効な ALTER パスワードを入力すると、EXECUTE ステートメントは実行されますが、DESCRIBE ステートメントは実行されません。

- バッチモードでは、EXECUTE ステートメントは処理されますが、DESCRIBE ステートメントは処理されません。



- 対話型モードでもバッチモードでも、無効な READ パスワードを指定すると、EXECUTE ステートメントは実行されません。
- コンパイル済み DATA ステッププログラムまたは DATA ステップビューに DESCRIBE ステートメントが含まれる場合には、ALTER パスワードが必要です。
- コンパイル済み DATA ステッププログラムまたは DATA ステップビューを置き換えるには、ALTER パスワードが必要です。

### 出力データセットを作成する(形式1)

1つまたは複数のデータセットを作成するには、DATA ステートメントを使用します。データセットオプションを使用すると、出力データセットをカスタマイズすることができます。次の DATA ステップでは、2つの出力データセット EXAMPLE1 および EXAMPLE2 を作成します。データセットオプション DROP を使用し、変数 IDNUMBER は EXAMPLE2 に書き込まないようにします。

```
data example1 example2 (drop=IDnumber);
  set sample;
  ...more SAS statements...
run;
```

### データセットを作成しない場合(形式2)

通常、DATA ステートメントでは、出力データセットの作成に使用するデータセット名を少なくとも1つ指定します。ただし、レポート出力または外部ファイルへのデータ出力を目的とする DATA ステップでは、出力データセットの作成は不要ことがあります。データセット名として \_NULL\_ キーワードを指定すると、オブザベーションをデータセットに書き込まずに、DATA ステップを実行できます。この例では、各オブザベーションの Name の値を SAS ログに書き込みます。出力データセットは作成されません。

```
data _NULL_;
  set sample;
  put Name ID;
run;
```

### DATA ステップビューの作成(形式3)

DATA ステップビューを先に作成して、後から実行することができます。次の DATA ステップの例では、DATA ステップビューが作成されます。また、SOURCE=ENCRYPT オプションを使用し、ソースコードの保存と暗号化を実行します。

```
data phone_list / view=phone_list (source=encrypt);
  set customer_list;
  ...more SAS statements...
run;
```

### コンパイル済み DATA ステッププログラムの作成(形式4)

DATA ステッププログラムをコンパイルして保存できるため、このストアドプログラムを後から実行できます。コンパイル済み DATA ステッププログラムにより、DATA ステッププログラムを繰り返しコンパイルする必要がなくなるため、処理コストを削減できます。次の DATA ステップの例では、DATA ステッププログラムのコンパイルと保存を行います。ここでは、ALTER パスワードオプションを使用します。そのため、ユーザーは既存のストアドプログラムを置き換えたり、コンパイル済みプログラムが置き換えられないように保護することができます。

```
data testfile / pgm=stored.test_program (alter=sales);
  set sales_data;
  ..more SAS statements...
run;
```

### DATA ステップビューの記述(形式5)

次の例では、DATA ステップビューで DESCRIBE ステートメントを使用し、ソースコードのコピーを SAS ログに書き込みます。

```
data view=inventory;
  describe;
run;
```

詳細については、“DESCRIBE ステートメント” (61 ページ)を参照してください。

### コンパイル済み DATA ステッププログラムの実行(形式6)

次の例では、コンパイル済みストア DATA ステッププログラムを実行しています。ここでは、DESCRIBE ステートメントを使用し、ソースコードのコピーを SAS ログに書き込みます。

```
libname stored 'SAS library';
data pgm=stored.employee_list;
  describe;
  execute;
run;

libname stored 'SAS library';
data pgm=stored.test_program;
  describe;
  execute;
  ..more SAS statements...
run;
```

詳細については、“DESCRIBE ステートメント” (61 ページ)および“EXECUTE ステートメント” (74 ページ)を参照してください。

## 例

### 例 1: 複数データファイルの作成とデータセットオプションの使用

次の DATA ステートメントでは、複数のデータセットを作成してから、出力データセットの内容を変更します。

```
data error (keep=subject date weight)
  fitness(label='Exercise Study'
  rename=(weight=pounds));
```

ERROR データセットには 3 つの変数が含まれています。FITNESS データセットにラベルを割り当てた後、変数の名前を *weight* から *pounds* に変更します。

### 例 2: 入力 DATA ステップビューの作成

次のデータステップでは、SAS データファイルのかわりに、入力 DATA ステップビューを作成します。

```
libname ourlib 'SAS-library';
data ourlib.test / view=ourlib.test;
  set ourlib.fittest;
```

```
tot=sum(of score1-score10);
run;
```

### 例 3: ビューおよびデータファイルの作成

次の DATA ステップでは、入力 DATA ステップビュー THEIRLIB.TEST を作成し、一時 SAS データセット SCORETOT を追加で作成します。

```
libname ourlib 'SAS-library-1';
libname theirlib 'SAS-library-2';
data theirlib.test scoretot
  / view=theirlib.test;
  set ourlib.fittest;
  tot=sum(of score1-score10);
run;
```

後続の DATA ステップまたは PROC ステップでビュー THEIRLIB.TEST が処理されるまで、データファイル SCORETOT は作成されません。

### 例 4: コンパイル済みプログラムの保存と実行

次に示す最初の DATA ステップでは、STORED.SALESFIG というコンパイル済みプログラムが作成されます。

```
libname in 'SAS-library-1';
libname stored 'SAS-library-2';
data salesdata / pgm=stored.salesfig;
  set in.sales;
  qtr1tot=jan+feb+mar;
run;
```

コンパイル済みプログラム STORED.SALESFIG を実行すると、データセット SALESDATA が作成されます。

```
data pgm=stored.salesfig;
run;
```

### 例 5: カスタムレポートの作成

次のプログラムの 2 番目の DATA ステップではカスタムレポートを生成します。また、\_NULL\_ キーワードが指定されているので、SAS データセットを作成せずに DATA ステップを実行します。

```
data sales;
  input dept : $10. jan feb mar;
  datalines;
shoes 4344 3555 2666
housewares 3777 4888 7999
appliances 53111 7122 41333
;
data _null_;
  set sales;
  qtr1tot=jan+feb+mar;
  put 'Total Quarterly Sales: '
    qtr1tot dollar12.;
run;
```

**例 6: コンパイル済み DATA ステッププログラムにパスワードを使用する**

次に示す最初の DATA ステップでは、STORED.ITEMS というコンパイル済み DATA ステッププログラムを作成します。また、このプログラムでは ALTER パスワードを使用しているため、プログラムへのアクセスが制限されます。

```
data sample;
  input Name $ TotalItems $;
  datalines;
Lin 328
Susan 433
Ken 156
Pat 340
;
proc print data=sample;
run;

libname stored 'SAS-library';

data employees / pgm=stored.items (alter=klondike);
  set sample;
  if TotalItems > 200 then output;
run;
```

この DATA ステップでは、コンパイル済み DATA ステッププログラム STORED.ITEMS を実行します。DESCRIBE ステートメントを使用し、ソースコードを SAS ログに出力します。このプログラムは ALTER パスワードを使用して作成されているため、DESCRIBE ステートメントを使用する場合はこのパスワードを使用する必要があります。パスワードを入力しないと、入力を求めるプロンプトが表示されます。

```
data pgm=stored.items (alter=klondike);
  describe;
  execute;
run;
```

**例 7: ネストレベルの表示**

次のプログラムには、2つのネストレベルが含まれています。4つのログメッセージが作成されます。このメッセージは、各ネストレベルの開始時と終了時に1つずつ作成されます。

```
data _null_ /nesting;
  do i = 1 to 10;
    do j = 1 to 5;
      put i=j;
    end;
  end;
run;
```

**ログ 2.2 ネストレベルのデバッグ(SAS ログから抜粋)**

```
6 data _null_ /nesting; 7 do i = 1 to 10; - 719 NOTE 719-185: *** DO begin level 1 ***.8 do j = 1 to 5; - 719
NOTE 719-185: *** DO begin level 2 ***.9 put i=j; 10 end; --- 720 NOTE 720-185: *** DO end level 2
***.11 end; --- 720 NOTE 720-185: *** DO end level 1 ***.12 run;
```

## 関連項目:

- [“DATA Step Programming in Hadoop” - SAS In-Database Products: User's Guide](#)
- [“DATA Step Programming in SAS LASR Analytic Server” - SAS LASR Analytic Server: Reference Guide](#)
- [“Definition of Data Set Options” \(SAS Viya Data Set Options: Reference\)](#)

## ステートメント:

- [“DESCRIBE ステートメント” \(61 ページ\)](#)
- [“EXECUTE ステートメント” \(74 ページ\)](#)
- [“LINK ステートメント” \(266 ページ\)](#)

---

## DATALINES ステートメント

データ行の始まりを示します。

**該当要素:** DATA ステップ

**カテゴリ:** ファイル操作

**種類:** 宣言

**別名:** CARDS, LINES

**制限事項:** このステートメントは CAS Server ではサポートされていません。

**参照項目:** データ行にセミコロンを含めることはできません。データにセミコロンが含まれている場合は、[“DATALINES4 ステートメント” \(59 ページ\)](#)を使用してください。

---

## 構文

**DATALINES;**

### 引数なし

外部ファイルに格納されているデータを読み込むのではなく、プログラムに直接入力するデータを読み込む場合は、DATALINES ステートメントを INPUT ステートメントとともに使用します。

## 詳細

### DATALINES ステートメントの使用

DATALINES ステートメントは DATA ステップの最後にあるステートメントです。このステートメントのすぐ後に最初のデータ行が続きます。入力データの最後を示すには、ヌルステートメント(1つのセミコロン)を使用します。

1つの DATA ステップでは、DATALINES ステートメントを1つのみ使用できません。複数のデータのセットを入力する場合は、個々に DATA ステップを使用します。

### 長いデータ行の読み込み

データ行の長さを扱うには、**CARDIMAGE** システムオプションを使用します。CARDIMAGE オプションを使用すると、空白が埋め込まれた 80 バイトのパンチカードイメージのようにデータ行を処理します。NOCARDIMAGE を使用すると、全体で 80 列よりも長いデータ行を処理します。

### インストリームデータでの入力オプションの使用

DATALINES ステートメントには、データの読み込みに使用できる入力オプションはありません。ただし、DATALINES ステートメントと INFILE ステートメントを組み合わせると、いくつかのオプションを使用できます。INFILE ステートメントに DATALINES ステートメントを指定してデータのソースを定義した後、必要なオプションを指定します。詳細については、“[例 2: オプションを使用したインストリームデータの読み込み](#)” (58 ページ) を参照してください。

### 比較

- データにセミコロンが含まれていない場合は、常に DATALINES ステートメントを使用します。データにセミコロンが含まれている場合は、DATALINES4 ステートメントを使用します。
- 次の SAS ステートメントを使用しても、データの読み込みやデータ格納先の指定を行えます。
  - INFILE ステートメントは、他のファイルに格納されている生データ行を指定します。INPUT ステートメントは、指定されたデータ行を読み込みます。
  - %INCLUDE ステートメントは、SAS プログラムステートメントや SAS ファイルまたは外部ファイルに格納されているデータ行を現在のプログラムに読み込みます。
  - SET、MERGE、MODIFY、UPDATE の各ステートメントは、既存の SAS データセットからオブザベーションを読み込みます。

### 例

#### 例 1: DATALINES ステートメントの使用

この例では、DATA ステップのオブザベーションごとに、データ行を読み込んだ後、2つの文字変数 NAME と DEPT に値を割り当てます。

```
data person;
  input name $ dept $;
  datalines;
John Sales
Mary Acctng
;
```

#### 例 2: オプションを使用したインストリームデータの読み込み

この例では、インストリームデータを読み込む INFILE ステートメントで使用可能なオプションを利用します。DELIMITER=オプションを使用すると、リスト入力を使用して、空白ではなくカンマで区切られたデータの値を読み込みます。

```
data person;
  infile datalines delimiter=';';
  input name $ dept $;
```

```
datalines;
John,Sales
Mary,Acctng
;
```

## 関連項目:

### ステートメント:

- “DATALINES4 ステートメント” (59 ページ)
- “INFILE ステートメント” (146 ページ)

### システムオプション:

- “CARDIMAGE System Option” (*SAS Viya System Options: Reference*)

---

## DATALINES4 ステートメント

セミコロンが含まれるデータ行の始まりを指定します。

<b>該当要素:</b>	DATA ステップ
<b>カテゴリ:</b>	ファイル操作
<b>種類:</b>	宣言
<b>別名:</b>	CARDS4, LINES4
<b>制限事項:</b>	このステートメントは CAS Server ではサポートされていません。

---

## 構文

**DATALINES4;**

### 引数なし

プログラムに直接入力するセミコロンを含むデータを読み込むには、DATALINES4 ステートメントを INPUT ステートメントとともに使用します。

## 詳細

DATALINES4 ステートメントは DATA ステップの最後にあるステートメントです。このステートメントのすぐ後に最初のデータ行が続きます。データ行の終わりには、第 1 列から第 4 列に配置される連続する 4 つのセミコロンを使用します。

## 比較

データにセミコロンが含まれている場合は、DATALINES4 ステートメントを使用します。データにセミコロンが含まれていない場合は、DATALINES ステートメントを使用します。

## 例: セミコロンが含まれるデータ行の読み込み

この例では、4つのセミコロンが検出されるまで、セミコロンが含まれるデータ行を読み込みます。プログラムは最後まで実行されます。

```

data biblio;
  input number citation $50.;
  datalines4;
  KIRK, 1988
  2 LIN ET AL., 1995; BRADY, 1993
  3 BERG, 1990; ROA, 1994; WILLIAMS, 1992
  ;;;

```

### 関連項目:

#### ステートメント:

- [“DATALINES ステートメント” \(57 ページ\)](#)

---

## DELETE ステートメント

現在のオブザベーションの処理を中止します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション  
CAS

**種類:** 実行

---

### 構文

**DELETE;**

#### 引数なし

DELETE を実行すると現在のオブザベーションはデータセットに書き込まれません。次の繰り返しを実行するために、すぐに DATA ステップの先頭に戻ります。

### 詳細

DELETE ステートメントは、IF-THEN ステートメントの THEN 句または条件を指定して実行される DO グループの一部としてよく使用されます。

### 比較

- DELETE ステートメントは、データセットからオブザベーションを除外する条件を設定するほうが簡単な場合、または現在のオブザベーションに対して DATA ステップの処理を継続する必要がない場合に使用します。
- オブザベーションを含める条件を設定するほうが簡単な場合は、サブセット化 IF ステートメントを使用します。
- DROP ステートメントと DELETE ステートメントを混同しないでください。DROP ステートメントは出力データセットから変数を除外します。DELETE ステートメントはオブザベーションを除外します。



## 例

### 例 1: IF-THEN ステートメントの一部として DELETE ステートメントを使用する

LEAFWT の値が欠損している場合、現在のオブザベーションを削除します。

```
if leafwt=. then delete;
```

### 例 2: DELETE ステートメントを使用して生データをサブセット化する

```
data topsales;
  infile file-specification;
  input region office product yrsales;
  if yrsales<100000 then delete;
run;
```

## 関連項目:

### ステートメント:

- [“DO ステートメント” \(62 ページ\)](#)
- [“DROP ステートメント” \(70 ページ\)](#)
- [“IF ステートメント、サブセット化” \(134 ページ\)](#)
- [“IF-THEN/ELSE ステートメント” \(137 ページ\)](#)

---

## DESCRIBE ステートメント

コンパイル済み DATA ステッププログラムまたは DATA ステップビューからソースコードを読み込みます。

**該当要素:** DATA ステップ

**カテゴリ:** アクション

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。  
DESCRIBE はコンパイル済み DATA ステッププログラムおよび DATA ステップビューにのみ使用します。

**要件** DATA ステートメントでは、PGM=オプションまたは VIEW=オプションを指定する必要があります。

## 構文

**DESCRIBE;**

### 引数なし

コンパイル済み DATA ステッププログラムまたは DATA ステップビューからプログラムソースコードを読み込む場合は、DESCRIBE ステートメントを使用します。SAS はソースステートメントを SAS ログに書き込みます。

## 詳細

コンパイル済み DATA ステッププログラムまたは DATA ステップビューからプログラムソースコードを読み込む場合は、EXECUTE ステートメントを使用せずに DESCRIBE ステートメントを使用します。ソースコードを読み込んでからコンパイル済み DATA ステッププログラムを実行する場合は、DESCRIBE ステートメントと EXECUTE ステートメントを使用します。DATA ステートメントでこれらのステートメントを使用する方法の詳細については、“[DATA ステートメント](#)” (47 ページ)を参照してください。

注: パスワード保護されたビューまたは DATA ステッププログラムを DESCRIBE するには、パスワードを指定しなければなりません。ビューまたはプログラムが複数のパスワード付きで作成されている場合、一番厳格なパスワードを指定します。他の SAS ファイルと同様に、ALTER パスワードは一番厳格で、READ パスワードは最も制限が緩いです。

## 関連項目:

### ステートメント:

- “[DATA ステートメント](#)” (47 ページ)
- “[EXECUTE ステートメント](#)” (74 ページ)

---

## DO ステートメント

1 単位として実行するステートメントのグループを指定します。

**該当要素:** DATA ステップ  
**カテゴリ:** CAS  
 制御  
**種類:** 実行

---

## 構文

```
DO;
...more SAS statements...
```

```
END;
```

### 引数なし

単純 DO グループの処理には、DO ステートメントを使用します。

## 詳細

DO ステートメントは、DO グループの処理を指定するもっとも簡単な形式です。DO ステートメントと END ステートメントではさまれたステートメントは、DO グループと呼ばれます。DO ステートメントは、DO グループ内でネストさせることができます。

注: システムのメモリサイズによっては、使用可能な DO ステートメントのネスト数が制限される場合があります。詳細については、SAS ドキュメントのシステムメモリでサポートされる DO ステートメントのネストレベルについての説明を参照してください。

通常、単純 DO ステートメントは、IF 条件が真か偽かに基づいてステートメントのグループの実行を指定する IF-THEN/ELSE ステートメントの中で使用されません。

## 比較

DO ステートメントは、他に 3 種類あります。

- 反復 DO ステートメントは、インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。反復 DO ステートメントには WHILE 句または UNTIL 句を含めることができます。
- DO UNTIL ステートメントは、DO ループを繰り返した後に条件を毎回確認して、条件が真になるまで DO ループ内のステートメントを実行します。
- DO WHILE ステートメントは、DO ループを繰り返す前に条件を毎回確認して、条件が真の間だけ DO ループ内のステートメントを実行します。

## 例: DO ステートメントの使用

この単純 DO グループでは、YEARS の値が 5 を超えている場合にのみ DO と END には含まれたステートメントが実行されます。YEARS の値が 5 以下の場合、DO グループのステートメントは実行されません。プログラムは、ELSE ステートメントの後に続く割り当てステートメントを実行します。

```
if years>5 then
do;
months=years*12;
put years= months=;
end;
else yrsleft=5-years;
```

## 関連項目:

### ステートメント:

- [“DO ステートメント、反復” \(63 ページ\)](#)
- [“DO UNTIL ステートメント” \(67 ページ\)](#)
- [“DO WHILE ステートメント” \(69 ページ\)](#)

---

## DO ステートメント、反復

インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 実行

---

## 構文

**DO** *index-variable*=*specification-1*<, ...*specification-n*>;  
...*more SAS statements*...

**END;**

## 引数

### *index-variable*

DO グループの実行を制御する値を持つ変数を指定します。

**ヒント** インデックス変数の削除を指定しない場合、作成されるデータセットにインデックス変数が含まれます。

**注意** **DO グループ内でインデックス変数を変更しないでください。** 反復 DO グループ内でインデックス変数を変更すると、無限ループが発生する場合があります。

### *specification*

1つの式または複数の式を次の形式で指定します。

*start* <TO *stop*> <BY *increment*> <WHILE(*expression*) | UNTIL(*expression*)>

初回実行時、DO グループは、*index-variable* が *start* に等しいとして実行されます。*start* の値は、ループの初回実行前に評価されます。

### *start*

インデックス変数の初期値を指定します。

TO *stop* または BY *increment* を指定せずに使用する場合、*start* の値は次の形式で指定されます。

*item-1* <, ...*item-n*>

*item* には、数値定数または文字定数を指定します。変数を指定する場合があります。文字定数は一重引用符で囲みます。リストに指定したそれぞれの値に対して DO グループが 1 度実行されます。WHILE 条件を追加する場合、この条件はそのすぐ前にある *item* にのみ適用されます。

**要件** TO *stop* または BY *increment* を使用する場合、*start* には数値または数値を生成する式を指定する必要があります。

**例** “例 1: さまざまな形式の反復 DO ステートメントを使用する” (66 ページ)

### TO *stop*

インデックス変数の終了値を指定します。

*start* と *stop* の両方を指定する場合、*index-variable* の値が *stop* になるまで、*increment* の値に基づいてステートメントが実行されます。*start* と *increment* のみを指定する場合、ステートメントによってループの中止が指示されるか、または DO ステートメントで指定された WHILE 式または UNTIL 式を満たすまで、*increment* の値に基づいてステートメントが実行されます。*stop* も *increment* も指定されない場合、このグループは *start* の値に基づいて実行されます。*stop* の値は、ループの初回実行前に評価されます。

**要件** *stop* の値は、数値または数値を生成する式を指定する必要があります。

**ヒント** DO グループでの *stop* の変更は、繰り返し回数に影響を与えることはありません。処理が終了する前にループの繰り返しを中止するには、*index-variable* の値を変更して *stop* の値を上回るようにするか、LEAVE ステートメントを使用してステートメントのループを中止します。

**例** “例 1: さまざまな形式の反復 DO ステートメントを使用する” (66 ページ)

### BY increment

*index-variable* の増分値を制御する正または負の数値(または数値を生成する式)を指定します。

*increment* の値はループを実行する前に評価されます。DO グループでの *increment* の変更は、繰り返し回数に影響を与えることはありません。*increment* を指定しない場合、インデックス変数は 1 つずつ増加します。*increment* が正の数値の場合、*start* の値を下限値にする必要があります。また、*stop* が存在する場合、この値はループの上限値にする必要があります。*increment* が負の数値の場合、*start* の値は上限値にする必要があります。また、*stop* が存在する場合、この値はループの下限値にする必要があります。

**例** “例 1: さまざまな形式の反復 DO ステートメントを使用する” (66 ページ)

### WHILE(expression) | UNTIL(expression)

DO グループの実行前または実行後に指定した SAS 式を評価します。式は丸かっこで囲みます。

WHILE 式はループの実行前に評価されるので、この式が真の間、グループ内のステートメントが繰り返し実行されます。UNTIL 式はループ実行後に評価されるので、この式が真になるまで、グループ内のステートメントが繰り返し実行されます。

**制限事項** WHILE または UNTIL の指定は、WHILE または UNTIL が指定されている句の最後の項目にのみ影響を与えます。

**参照項目:** “DO WHILE ステートメント” (69 ページ)および“DO UNTIL ステートメント” (67 ページ)

**例** “例 1: さまざまな形式の反復 DO ステートメントを使用する” (66 ページ)

**要件** 反復 DO ステートメントには、少なくとも 1 つの *specification* 引数が必要です。

**ヒント** オプションである TO 句と BY 句の位置は逆にすることができます。

複数の *specification* を使用する場合、それぞれの *specification* が実行前に評価されます。

## 比較

DO ステートメントは、他に 3 種類あります。

- DO ステートメントは、DO グループの処理を指定するもっとも簡単な形式です。通常、IF-THEN/ELSE ステートメントの中で使用され、1 単位として実行するステートメントのグループを指定します。
- DO UNTIL ステートメントは、DO ループを繰り返した後に条件を毎回確認して、条件が真になるまで DO ループ内のステートメントを実行します。
- DO WHILE ステートメントは、DO ループを繰り返す前に条件を毎回確認して、条件が真の間だけ DO ループ内のステートメントを実行します。

## 例

### 例 1: さまざまな形式の反復 DO ステートメントを使用する

- 次の反復 DO ステートメントでは、*start* の値に項目のリストを使用します。
  - `do month='JAN','FEB','MAR';`
  - `do count=2,3,5,7,11,13,17;`
  - `do i=5;`
  - `do i=var1, var2, var3;`
  - `do i='01JAN2001'd,'25FEB2001'd,'18APR2001'd;`
- 次の反復 DO ステートメントでは、*start TO stop* 構文を使用します。
  - `do i=1 to 10;`
  - `do i=1 to exit;`
  - `do i=1 to x-5;`
  - `do i=1 to k-1, k+1 to n;`
  - `do i=k+1 to n-1;`
- 次の反復 DO ステートメントでは、*BY increment* 構文を使用します。
  - `do i=n to 1 by -1;`
  - `do i=.1 to .9 by .1, 1 to 10 by 1, 20 to 100 by 10;`
  - `do count=2 to 8 by 2;`
- 次の反復 DO ステートメントでは、WHILE 句と UNTIL 句を使用します。
  - `do i=1 to 10 while(x<y);`
  - `do i=2 to 20 by 2 until((x/3)>y);`
  - `do i=10 to 0 by -1 while(month='JAN');`
- この例では、I=1 および I=2 の場合に DO ループが実行されます。I=3 の場合に WHILE 条件が評価され、この条件が真の場合に DO ループが実行されません。
 

```
DO I=1,2,3 WHILE (condition);
```

### 例 2: 無限ループを発生させずに反復 DO ステートメントを使用する

次のどちらの例でも DO グループを 10 回実行します。最初の例では推奨方法を示します。

```
/* correct coding */
```

```
do i=1 to 10;
  ...more SAS statements...
end;
```

次の例では、TO 引数および BY 引数を使用します。

```
do i=1 to n by m;
  ...more SAS statements...
  if i=10 then leave;
end;
if i=10 then put 'EXITED LOOP';
```

### 例 3: DO ループの実行を中止する

この例では、ループを終了させるために、インデックス変数の値を EXIT の現在値に設定します。

```
data iterate1;
  input x;
  exit=10;
  do i=1 to exit;
    y=x*normal(0);
    /* if y>25,      */
    /* changing i's value */
    /* stops execution */
    if y>25 then i=exit;
  output;
end;
datalines;
5
000
2500
;
```

### 関連項目:

#### ステートメント:

- [“ARRAY ステートメント” \(20 ページ\)](#)
- [“配列参照ステートメント” \(25 ページ\)](#)
- [“DO ステートメント” \(62 ページ\)](#)
- [“DO UNTIL ステートメント” \(67 ページ\)](#)
- [“DO WHILE ステートメント” \(69 ページ\)](#)
- [“GO TO ステートメント” \(132 ページ\)](#)

---

## DO UNTIL ステートメント

条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 実行

## 構文

```
DO UNTIL (expression);  
...more SAS statements...
```

```
END;
```

## 引数

### (*expression*)

SAS 式を丸かっこで囲んで指定します。少なくとも 1 つの *expression* を指定する必要があります。

## 詳細

式は、DO ループのステートメントが実行されたループの最後に評価されます。この式が真になると、DO ループは実行されません。

注: 少なくとも 1 度は DO ループが実行されることになります。

## 比較

DO ステートメントは、他に 3 種類あります。

- DO ステートメントは、DO グループの処理を指定するもっとも簡単な形式です。通常、IF-THEN/ELSE ステートメントの中で使用され、1 単位として実行するステートメントのグループを指定します。
- 反復 DO ステートメントは、インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。
- DO WHILE ステートメントは、DO ループを繰り返す前に条件を毎回確認して、条件が真の間だけ DO ループ内のステートメントを実行します。DO UNTIL ステートメントではループの最後に条件を評価するのに対し、DO WHILE ステートメントではループの最初に条件を評価します。

注: DO UNTIL ループのステートメントは少なくとも 1 度は実行されますが、DO WHILE ループのステートメントは条件が偽の場合には 1 度も実行されません。

## 例: DO UNTIL ステートメントを使用したループの繰り返し

次のステートメントでは、N の値が 5 以上になるまでループが繰り返されます。式  $N \geq 5$  はループの最後に評価されます。(N の値が 0、1、2、3、4 の)全部で 5 回繰り返しが実行されます。

```
n=0;  
do until(n>=5);  
  put n=;  
  n+1;  
end;
```

## 関連項目:

ステートメント:



- “DO ステートメント” (62 ページ)
- “DO ステートメント、反復” (63 ページ)
- “DO WHILE ステートメント” (69 ページ)

---

## DO WHILE ステートメント

条件が真の間、DO ループ内のステートメントを繰り返し実行します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 実行

---

### 構文

```
DO WHILE (expression);
...more SAS statements...
```

```
END;
```

### 引数

**(*expression*)**

SAS 式を丸かっこで囲んで指定します。少なくとも 1 つの *expression* を指定する必要があります。

### 詳細

式は、ループの最初に DO ループのステートメントの実行前に評価されます。この式が真の場合、DO ループは繰り返し実行されます。この式が最初の評価時に偽となる場合、DO ループは 1 度も実行されません。

### 比較

DO ステートメントは、他に 3 種類あります。

- DO ステートメントは、DO グループの処理を指定するもっとも簡単な形式です。通常、IF-THEN/ELSE ステートメントの中で使用され、1 単位として実行するステートメントのグループを指定します。
- 反復 DO ステートメントは、インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。
- DO UNTIL ステートメントは、DO ループを繰り返した後に条件を毎回確認して、条件が真になるまで DO ループ内のステートメントを実行します。DO WHILE ステートメントではループの最初に条件を評価するのに対し、DO UNTIL ステートメントではループの最後に条件を評価します。

**注:** 式の評価結果が偽の場合、DO WHILE ループのステートメントは実行されません。ただし、DO UNTIL の式はループの最後に評価されるため、DO UNTIL ループのステートメントは少なくとも 1 度実行されます。

## 例: DO WHILE ステートメントの使用

次のステートメントでは、N の値が 5 未満の間ループが繰り返されます。式  $N < 5$  はループの最初に評価されます。(N の値が 0、1、2、3、4 の)全部で 5 回繰り返しが実行されます。

```
n=0;
do while(n<5);
  put n=;
  n+1;
end;
```

### 関連項目:

#### ステートメント:

- “DO ステートメント” (62 ページ)
- “DO ステートメント、反復” (63 ページ)
- “DO UNTIL ステートメント” (67 ページ)

---

## DROP ステートメント

出力 SAS データセットから変数を除外します。

**該当要素:** DATA ステップ

**カテゴリ:** 情報  
CAS

**種類:** 宣言

---

### 構文

**DROP** *variable-list*;

### 引数

#### *variable-list*

出力データセットから除外する変数の名前を指定します。

**ヒント** SAS で使用可能な任意の形式で変数をリストできます。

---

### 詳細

DROP ステートメントは、同一の DATA ステップ内で作成されたすべての SAS データセットに適用されます。また、ステップのどの位置に指定してもかまいません。DROP ステートメントの変数は DATA ステップでの処理が可能です。DROP または KEEP ステートメントが存在しない場合、DATA ステップで作成されたすべてのデータセットにすべての変数が含まれます。同一の DATA ステップで DROP ステートメントと KEEP ステートメントの両方を使用しないでください。

## 比較

- DROP ステートメントと DROP=データセットオプションの違いは次の点になります。
  - SAS プロシジャステップでは、DROP ステートメントを使用できません。
  - DROP ステートメントは DATA ステートメントで指定したすべての出力データセットに適用されます。一部のデータセットから変数を除外し、その他のデータセットから除外しない場合は、DATA ステートメントで DROP=データセットオプションを使用します。
- KEEP ステートメントは、出力データセットへ書き込む変数のリストを指定する同種のステートメントです。含める変数が除外する変数よりも著しく少ない場合、DROP ステートメントのかわりに KEEP ステートメントを使用します。
- DROP ステートメントと DELETE ステートメントを混同しないでください。DROP ステートメントは出力データセットから変数を除外します。DELETE ステートメントはオブザベーションを除外します。

## 例

### 例 1: DROP ステートメントの基本的な使用法

これらの例では、DROP ステートメントで変数をリストする場合の正しい構文を示します。

- drop time shift batchnum;
- drop grade1-grade20;

### 例 2: 出力データセットから変数を除外する

この例では、変数 PURCHASE と REPAIR が処理で使用されていますが、出力データセットである INVENTORY には書き込まれません。

```
data inventory;
  drop purchase repair;
  infile file-specification;
  input unit part purchase repair;
  totcost=sum(purchase,repair);
run;
```

## 関連項目:

### データセットオプション:

- [“DROP= Data Set Option” \(SAS Viya Data Set Options: Reference\)](#)

### ステートメント:

- [“DELETE ステートメント” \(60 ページ\)](#)
- [“KEEP ステートメント” \(216 ページ\)](#)

---

## END ステートメント

DO グループまたは SELECT グループの処理を終了します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 宣言

---

### 構文

**END;**

### 引数なし

DO グループまたは SELECT グループの処理を終了するには END を使用します。

### 詳細

END ステートメントは、DO グループまたは SELECT グループ内の最後のステートメントとして指定する必要があります。

### 例: END ステートメントの使用

単純 DO グループと単純 SELECT グループの例を次に示します。

- do;  
  ...more SAS statements...  
end;
- select(expression);  
  when(expression) SAS statement;  
  otherwise SAS statement;  
end;

### 関連項目:

#### ステートメント:

- [“DO ステートメント” \(62 ページ\)](#)
- [“SELECT ステートメント” \(375 ページ\)](#)

---

## ENDSAS ステートメント

現在の DATA ステップまたは PROC ステップを実行した後に、SAS ジョブまたは SAS セッションを終了します。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

## 構文

**ENDSAS;**

### 引数なし

ENDSAS ステートメントは、SAS ジョブまたは SAS セッションを終了します。

### 詳細

ENDSAS は対話型セッションで使用できます。

注: DATA ステップ内で ENDSAS ステートメントが見つかった時点で実行されます。ABORT RETURN ステートメントを使用して、(IF-THEN ステートメントの句や SELECT ステートメントの句などで) エラー条件が発生した場合に処理を中止します。

---

## ERROR ステートメント

`_ERROR_` を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。

**該当要素:** DATA ステップ

**カテゴリ:** アクション

CAS

**種類:** 実行

---

## 構文

**ERROR** <*message*>;

### 引数なし

引数を指定せずに ERROR を使用する場合は、自動変数 `_ERROR_` が 1 に設定され、ブランクのメッセージがログに書き込まれます。

### 引数

*message*

メッセージをログに書き込みます。

ヒント *message* には、文字リテラル(一重引用符で囲む)、変数名、出力形式、ポインタコントロールを使用できます。

---

### 詳細

ERROR ステートメントは自動変数 `_ERROR_` を 1 に設定します。SAS ログへの指定したメッセージの書き込みはオプションです。`_ERROR_ =1` の場合、現在のオブザベーションに対応するデータ行を SAS ログに書き込みます。

ERROR ステートメントは、次のステートメントの組み合わせに相当します。

- `_ERROR_` を 1 に設定する割り当てステートメント
- FILE LOG ステートメント

- PUT ステートメント(メッセージを指定する場合)
- PUT; ステートメント(メッセージを指定しない場合)
- FILE をあらかじめ指定した設定にリセットする別の FILE ステートメント

### 例: エラーメッセージの書き込み

次の両方の例とも、IF-THEN ステートメントの条件を満たすオブザベーションごとに、エラーメッセージ、変数名、値をログに書き込みます。

- この例では、ERROR ステートメントは FILE ステートメントの指定をあらかじめ指定した設定に自動的にリセットします。

```
file file-specification;
  if type='teen' & age > 19 then
    error 'type and age don't match ' age=;
```

- この例では、ERROR ステートメントのかわりに、複数のステートメントを使用して同じ結果を出します。

```
file file-specification;
  if type='teen' & age > 19 then
  do;
    file log;
    put 'type and age don't match ' age=;
    _error_=1;
  file file-specification;
  end;
```

### 関連項目:

#### ステートメント:

- [“PUT ステートメント” \(318 ページ\)](#)

---

## EXECUTE ステートメント

コンパイル済み DATA ステッププログラムを実行します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。  
EXECUTE はコンパイル済み DATA ステッププログラムのみで使用します。

**要件** DATA ステップ内で PGM=オプションを指定します。

### 構文

**EXECUTE;**

## 引数なし

EXECUTE ステートメントは、コンパイル済み DATA ステッププログラムを実行します。

## 詳細

ソースコードを読み込んでからコンパイル済み DATA ステッププログラムを実行する場合は、同じ DATA ステップ内で EXECUTE ステートメントと DESCRIBE ステートメントを使用します。どちらのステートメントであるかを指定しない場合、EXECUTE ステートメントであるとみなされます。ステートメントを使用する順番は入れ替えることができます。DATA ステッププログラムは、ステップ境界に到達すると実行されます。DATA ステートメントでこれらのステートメントを使用する方法の詳細については、“[DATA ステートメント](#)” (47 ページ)を参照してください。

## 関連項目:

### ステートメント:

- “[DATA ステートメント](#)” (47 ページ)
- “[DESCRIBE ステートメント](#)” (61 ページ)

---

## FILE ステートメント

PUT ステートメントの現在の出力ファイルを指定します。

**該当要素:** DATA ステップ

**カテゴリ:** ファイル操作

CAS

**種類:** 実行

**制限事項:** LOG は、CAS サーバーでのみ使用可能な *file-specification* です。

*device-type* 引数は、CAS サーバーでは使用できません。

SAS がロックダウン状態にある場合、ロックダウンパスリストに含まれていないファイルに関しては、FILENAME ステートメントを使用できません。

## 構文

```
FILE file-specification <device-type> <PERMISSION='permission-value'>
<ENCODING='encoding-value' >
<options>;
```

## 必須引数

### *file-specification*

DATA ステップの PUT ステートメントの出力先に使用する外部ファイルを指定します。*file-specification* には、次の形式を指定できます。

### 'external-file'

外部ファイルの物理名を一重引用符で囲んで指定します。物理名には動作環境でファイルを判別できる名前を指定します。

***fileref***

外部ファイルのファイル参照名を指定します。

**制限事項** この引数は CAS Server ではサポートされていません。

**要件** *fileref* は、FILENAME ステートメントまたは FILENAME 関数を使用するか、適切な動作環境のコマンドを使用して、あらかじめ前のステップで外部ファイルに関連付けておく必要があります。実行時に *fileref* を割り当てる唯一の方法は、FILE ステートメントで FILEVAR=オプションを使用することです。

**参照項目:** [“FILENAME ステートメント” \(96 ページ\)](#)

***fileref(file)***

ファイルの集約記憶域を表す外部ファイルに事前に割り当てられているファイル参照名を指定します。ファイル参照名の後ろに続けて、対応する集約記憶域に含まれているファイル名またはメンバ名をカッコで囲んで指定します。

**制限事項** この引数は CAS Server ではサポートされていません。

**要件** *fileref* は、FILENAME ステートメントまたは FILENAME 関数を使用するか、適切な動作環境のコマンドを使用して、事前に外部ファイルに関連付けておく必要があります。

**注** 集約記憶域に格納されているファイルの名前が有効な SAS 名ではない場合は、その名前を引用符で囲む必要があります。

**参照項目:** [“FILENAME ステートメント” \(96 ページ\)](#)

**LOG**

予約済みのファイル参照名です。PUT ステートメントで生成される出力を SAS ログに表示するように指示します。

DATA ステップの実行を開始するたびに、PUT ステートメントの出力先を指定するファイル参照名を自動的に LOG に設定します。そのため、FILE ステートメントで別の出力先を指定しない限り、DATA ステップの最初の PUT ステートメントで生成される出力は、必ず SAS ログに出力されます。

**ヒント** デフォルトでは、出力行は SAS ログに書き込まれます。そのため、FILE LOG ステートメントを使用するのは、変更した出力先をデフォルトの設定へ戻す場合、他の FILE ステートメントオプションを指定する場合です。

**PRINT**

予約済みのファイル参照名です。PUT ステートメントで生成される出力を、SAS プロシジャで生成される出力と同じファイルに出力するように指示します。

**制限事項** この引数は CAS Server ではサポートされていません。

**操作** ファイルへ出力する場合、N=オプションの値は 1 または PAGESIZE とする必要があります。



**動作環境** ファイルに出力されるキャリッジコントロール文字は、動作環境によって異なる場合があります。

**ヒント** ファイル参照名に PRINT を指定した場合、キャリッジコントロール文字を使用して、出力ファイルの特性にあった出力を行います。

**ヒント** file-specification に指定したディレクトリにファイルが存在しない場合、ファイルが作成されます。file-specification に指定したディレクトリが存在しない場合、SYSERR マクロ変数が設定されます。この変数は、ERRORCHECK オプションが STRICT に設定されているかどうかを調べます。

## オプション引数

### device-type

デバイスの種類またはアクセス方式を指定します。これは、ファイル参照名が入出力デバイスや物理ファイルではない場所を参照している場合に使用されます。

#### ACTIVEMQ

このアクセス方式を指定すると、ActiveMQ メッセージブローカーへアクセスできます。

**制限事項** この引数は CAS Server ではサポートされていません。

**操作** DATA ステップでアクセス方式のオプションを認識できない場合、DATA ステップはこのオプションをアクセス方式に渡して処理します。

#### DISK

デバイスがディスクドライブであると指定します。

**制限事項** この引数は CAS Server ではサポートされていません。

**ヒント** ディスク上のファイルにファイル参照名を割り当てる場合、DISK を指定する必要はありません。

#### DUMMY

ファイルへの出力を破棄するように指定します。

**制限事項** この引数は CAS Server ではサポートされていません。

**ヒント** テストを実行する場合は DUMMY を指定すると便利です。

#### GTERM

出力デバイスの種類がグラフィックデータを受信するグラフィックデバイスであると指定します。

**制限事項** この引数は CAS Server ではサポートされていません。

#### JMS

Java Message Service (JMS) の送信先を指定します。

**制限事項** この引数は CAS Server ではサポートされていません。

**PIPE**

名前の付いていないパイプを指定します。

制限事項 この引数は CAS Server ではサポートされていません。

---

**PLOTTER**

バッファなしのグラフィック出力デバイスを指定します。

制限事項 この引数は CAS Server ではサポートされていません。

---

**PRINTER**

プリンタまたはプリンタプールファイルを指定します。

制限事項 この引数は CAS Server ではサポートされていません。

---

**SFTP**

SFTP アクセス方式を指定します。

制限事項 この引数は CAS Server ではサポートされていません。

---

操作 DATA ステップでアクセス方式のオプションを認識できない場合、DATA ステップはこのオプションをアクセス方式に渡して処理します。

---

参照項目 SFTP アクセス方式で指定可能なオプションの一覧については、[“FILENAME ステートメント、SFTP アクセス方式” \(109 ページ\)](#)を参照してください。

---

**TAPE**

テープドライブを指定します。

制限事項 この引数は CAS Server ではサポートされていません。

---

**TEMP**

ファイル名が割り当てられている間だけ存在する一時ファイルを作成します。この一時ファイルは論理名からのみアクセスできます。また、論理名が存在する間だけ使用できます。

制限事項 この引数は CAS Server ではサポートされていません。

---

物理パス名は指定しないでください。物理パス名を指定するとエラーが発生します。

---

ヒント TEMP デバイスで操作するファイルは、DISK ファイルに対して、同じ属性を保有し、同じように動作します。

---

**TERMINAL**

ユーザーの端末を指定します。

制限事項 この引数は CAS Server ではサポートされていません。

---

**UPRINTER**

ユニバーサル印刷プリンタの定義名を指定します。

制限事項 この引数は CAS Server ではサポートされていません。

---

**ヒント** FILENAME ステートメントにプリンタ名を指定しない場合、PRINTERPATH オプションによって、使用するユニバーサルプリンタと出力先が制御されます。

## URL

URL アクセス方式を指定します。

**制限事項** この引数は CAS Server ではサポートされていません。

**操作** DATA ステップでアクセス方式のオプションを認識できない場合、DATA ステップはこのオプションをアクセス方式に渡して処理します。

**参照項目:** URL アクセス方式で指定可能なオプションの一覧については、“[FILENAME ステートメント、URL アクセス方式](#)” (115 ページ)を参照してください。

**別名** DEVICE=*device-type*

**デフォルト** DISK

**要件** *device-type* または DEVICE=*device-type* は、ステートメントの *file-specification* の直後に記述する必要があります。

## ENCODING='encoding-value'

出力ファイルへの書き込み時に使用するエンコーディングを指定します。ENCODING=の値は、出力ファイルのエンコーディングが現在のセッションエンコーディングとは異なることを示しています。

出力ファイルにデータを書き込む場合は、セッションエンコーディングから指定したエンコーディングにデータがトランスコードされます。

エンコーディングの有効な値については、“[Overview to SAS Language Elements That Use Encoding Values](#)” (*SAS Viya National Language Support: Reference Guide*)を参照してください。

## PERMISSION='permission-value'

指定ファイル参照名に対して設定する権限を指定します。2 つ以上の権限値セットを指定するには、引用符内でカンマを使用して区切ります。

次の形式で *permission-value* を指定します。

**A::*trustee\_type*::<permissions>**

'A'は、これらがアクセス権限であることを示します。現在サポートされている値は他にありません。

*trustee\_type* が取り得る値を次に示します。

- u ユーザー
- g グループ(ファイルのグループ所有者)
- o その他(他のユーザーすべて)

権限値は、英文字 *r* (読み取り)、*w* (書き込み)、および *x* (実行)の値をこの順序で取ります。これらの権限の 1 つを付与しない場合は、その位置に '-' を入力します(たとえば、*r-x* や *rw-* など)。

ファイル参照名に対して読み取り、書き込み、および実行権限を得たいとします。また、ファイルのグループ所有者に対して読み取りおよび実行権限を

指定するとします。最後に、他のユーザーすべてがファイルに対して読み取り権限のみを得ることを許可します。これらのオプションは、次のように指定できます。

```
permission='A::u::rwx,A::g::r-x,A::o::r--'
```

3つのトラスティの種類すべてに対して権限値を指定します。権限値のリストで省略したトラスティの種類はいずれも、指定ファイル参照名へのアクセスをすべて拒否されます。たとえば、次の権限値を使用したとします。

```
permission='A::u::rwx,A::g::r-x'
```

この場合、指定ファイルにアクセスできるのは所有者およびグループ所有者のみになります。所有者およびグループ所有者以外のユーザーはいずれもファイルへのアクセスをすべて拒否されます。

## オプション

### **BLKSIZE=***block-size*

1回のI/O操作で物理的に書き込まれるバイト数を指定します。デフォルト値は8Kです。最大値は1G-1です。

別名 BLK=

### **COLUMN=***variable*

変数名を指定します。指定した変数の値は、ポインタの現在の列位置を示す値に設定されます。この変数は、自動変数と同じようにデータセットに書き込まれません。

別名 COL=

参照項目: [LINE= \(84 ページ\)](#)

### **DELIMITER=** *delimiter(s)*

リスト出力の *delimiter* に使用する、ブランクのかわりの区切り文字を指定します。使用できる区切り文字は次の通りです。

#### **'list-of-delimiting-characters'**

区切り文字として出力する文字を1つまたは複数指定します。

要件 指定する区切り文字のリストは引用符で囲む必要があります。

#### ***character-variable***

文字変数の名前を指定します。指定した変数の値が区切り文字として使用されます。

別名 DLM=

デフォルト ブランク

制限事項 文字列や文字変数を指定することはできますが、区切り文字として使用されるのは、指定した文字列や変数の最初の1文字だけです。FILE DLM=の処理は、INFIL DELIMITER=の処理とは異なります。

操作 出力に区切り文字を含める場合は、区切り文字を区別するDSDオプションを指定する必要があります。

ヒント DELIMITER=オプションはコロン(:)修飾子と併用できます(修飾リスト出力)。

区切り文字では大文字と小文字が区別されます。

参照項目: [“DLMSTR= delimiter” \(81 ページ\)](#)および[“DSD \(delimiter sensitive data\)” \(82 ページ\)](#)

### **DLMSTOPT= 'T' |'t'**

DLMSTR=T オプションの解析オプションを指定します。指定すると、区切り文字の末尾にある空白を削除します。

要件 DLMSTOPT=T オプションは、DLMSTR=オプションを使用している場合にのみ有効です。

ヒント DLMSTOPT=T オプションは、区切り文字として変数を使用する場合に便利です。

参照項目: [DLMSTR= \(81 ページ\)](#)

### **DLMSTR= delimiter**

リスト出力の *delimiter* (空白のかわりの区切り文字) として使用する文字列を指定します。使用できる区切り文字は次の通りです。

#### **'delimiting-string'**

区切り文字として出力する文字列を指定します。

要件 文字列は一重引用符で囲みます。

#### **character-variable**

文字変数の名前を指定します。指定した変数の値が区切り文字として使用されます。

デフォルト  
空白

操作 FILE ステートメントに DLMSTR=オプションを複数指定すると、最後に指定した DLMSTR=オプションが使用されます。DELIMITER=オプションと DLMSTR=オプションの両方を指定すると、最後に指定したオプションが使用されます。

RECFM=N を指定する場合は、サイズが大きい入力項目でも十分に保持できる値が LRECL に指定されていることを確認してください。指定した値が十分ではない場合、区切り文字によってレコード境界間で分割される場合があります。

参照項目: [DELIMITER= \(80 ページ\)](#)、[DLMSTOPT= \(81 ページ\)](#)、および [DSD \(82 ページ\)](#) DROPOVER

### **DROPOVER**

FILE ステートメントの LINESIZE=オプションまたは LRECL=オプションに指定した出力行の長さを超えるデータを切り捨てます。

デフォルトでは、現在の行の長さを超えるデータは、新しい行に書き込まれます。一方、DROPOVER を指定すると、現在の行に新しいデータ項目を書き出すスペースがなければ、そのデータ項目全体が切り捨てられます(または無視されます)。項目全体が切り捨てられる場合、カラムポインタは、現在の行

の最後の値を書き出した位置に留まります。十分なスペースが残っているか、カラムポインタの位置を元に戻すと、PUT ステートメントで別の項目を現在の行に書き出せます。データ項目が切り捨てられても、DATA ステップの実行は正常に続行されます(自動変数\_ERROR\_=0)。DATA ステップの最後に、どのデータが切り捨てられたかを示すメッセージが各ファイルに対して出力されます。

デフォルト FLOWOVER

ヒント DROPOVER オプションは、PUT ステートメントで現在指定されている行の長さを超えて書き込みを行おうとした場合に、現在の長さを超えたデータ項目を新しい行に出力しない場合に使用します。

参照項目: ["FLOWOVER" \(83 ページ\)](#)および["STOPOVER" \(89 ページ\)](#)

### DSD (delimiter sensitive data)

タブやカンマなどの区切り文字を含むデータ値を一重引用符で囲むように指定します。DSD オプションでは、区切り文字を含むデータ値をリスト出力に書き込むことができます。このオプションは他の種類の出力(例:フォーマット出力、カラム出力、名前付き出力)では無視されます。データ値に含まれる二重引用符は2回繰り返されます。変数の値に区切り文字が含まれ、FILE ステートメントに DSD を指定する場合、変数の値は出力が生成されるときに二重引用符で囲まれます。例として、次のコードを示します。

```
DATA _NULL_;
  FILE log dsd;
  x="lions, tigers, and bears";
  put x ' "Oh, my!";
run;
```

出力結果は次のようになります。

```
""lions, tigers, and bears""", "Oh, my!"
```

引用符で囲まれている(テキスト)文字列に区切り文字が含まれ、FILE ステートメントに DSD が指定されている場合、引用符を含む文字列は PUT ステートメントを実行するときに二重引用符で囲まれません。例として、次のコードを示します。

```
DATA _NULL_;
  FILE log dsd;
  PUT 'lions, tigers, and bears';
run;
```

出力結果は次のようになります。

```
lions, tigers, and bears
```

操作 DSD を指定する場合、デフォルトの区切り文字はカンマ(,)です。別の区切り文字を使用する場合は、DELIMITER=オプションまたは DLMSTR=オプションを指定します。

ヒント デフォルトでは、指定した区切り文字を含まないデータ値は引用符で囲まれません。ただし、チルダ(~)修飾子を使用すると、欠損値を含むすべてのデータ値が引用符で囲まれます。その場合、区切り文字を含まないデータ値も引用符で囲まれます。

参照 DELIMITER= (80 ページ)および DLMSTR= (81 ページ)  
項目:

### FILENAME=*variable*

文字変数の名前を指定します。指定した変数の値には、PUT ステートメントの出力で現在開いているファイルの物理名が設定されます。物理名には動作環境でファイルを判別できる名前を指定します。

ヒント この変数は、自動変数と同じようにデータセットに書き込まれません。

文字変数のデフォルトの長さは 8 文字です。変数の長さが 8 バイトよりも長い場合、LENGTH ステートメントを使用して物理ファイル名の値を格納できるように十分な長さに設定してください。

参照項 FILEVAR= (83 ページ)  
目:

例 “例 4: 現在の出力ファイルの識別” (94 ページ)

### FILEVAR=*variable*

変数の名前を指定します。この変数の値が変化すると、FILE ステートメントは現在の出力ファイルを閉じます。FILE ステートメントを次に実行するとき、新しいファイルを開きます。次の PUT ステートメントの出力先は、FILEVAR=オプションの変数の値に指定した新しいファイルになります。

制限 FILEVAR=オプションの変数の値には、物理ファイル名を含む文字列を  
事項 指定します。

操作 FILEVAR=オプションを使用すると、*file-specification* は、実際のファイル名またはファイルに対して事前に割り当てられたファイル参照名ではなく、プレースホルダになります。このプレースホルダを使用して、処理情報を SAS ログに出力します。プレースホルダには、ファイル参照名と同じルールが適用されます。

ヒント この変数は、自動変数と同じようにデータセットに書き込まれません。

文字変数のデフォルトの長さは 8 文字です。8 文字を超える物理ファイル名がある場合、LENGTH ステートメントや INPUT ステートメントなどの他のステートメントを使用して、FILEVAR=オプションの変数に十分な長さを割り当ててください。

参照 FILENAME= (83 ページ)  
項目:

例 “例 5: 現在の出力ファイルを動的に変更する” (94 ページ)

### FLOWOVER

現在の行の長さを超えるデータは、新しい行に書き込まれます。指定した行の最大長(FILE ステートメントの LINESIZE=オプションに指定)を超える書き込みを PUT ステートメントが行う場合、現在の出力行をファイルに出力した後、現在の行の長さを超えるデータ項目は新しい行に書き込まれます。

デフォルト FLOWOVER



**操作** PUT ステートメントに後置@が含まれる場合、ポインタは新しい行のデータ項目を出力した後の位置で停止します。そのため、次のPUT ステートメントは、同じ行に書き込むことができます。このプロセスは、入力データの最後に到達するか、または後置@を指定しないPUT ステートメントによって現在の行がファイルに書き込まれるまで続きます。

**参照項目:** [“DROPOVER” \(81 ページ\)](#)および[“STOPOVER” \(89 ページ\)](#)

### FOOTNOTES | NOFOOTNOTES

現在定義されているフットノートを出力するかどうかを制御します。

**別名** FOOTNOTE | NOFOOTNOTE

**デフォルト** NOFOOTNOTES

**要件** DATA ステップで出力するレポートにフットノートを出力するには、FILE ステートメントに FOOTNOTE オプションを指定する必要があります。

### HEADER=*label*

ステートメントラベルを定義します。このラベルは、新しいページへの出力を開始するたびに実行する SAS ステートメントのグループを特定します。

**制限事項** ラベルの後ろに記述する最初のステートメントは、実行ステートメントにする必要があります。それ以降は、任意のステートメントを使用できます。

HEADER=オプションは、出力ファイルに書き込む場合や PRINT=オプションを含める場合にのみ使用します。

**ヒント** このグループのステートメントが DATA ステップの反復のたびに実行されるのを防ぐには、RETURN ステートメントを 2 つ使用します。1 つはラベルの前に記述し、もう 1 つはグループの最後のステートメントとして記述します。

**例** [“例 1: 新規ページの開始時にステートメントを実行する” \(92 ページ\)](#)

### LINE=*variable*

変数の名前を指定します。ここに指定した変数の値には、出力ポインタを移動できる行範囲での現在の相対行番号が設定されます。変数名を指定すると、値が自動的に割り当てられます。

**範囲** 1 から N=オプションまたは#*n* 行ポインタコントロールに指定した値。どちらも指定されていない場合、LINE=変数の値は 1 になります。

**ヒント** この変数は、自動変数と同じようにデータセットに書き込まれません。

LINE=に指定した変数の値は、PUT ステートメントの実行の最後に、次に出力できる行の番号に対して設定されます。

### LINESIZE=*line-size*

レポートの 1 行あたりの最大列数、およびデータファイルの最大レコード長を設定します。



別名	LS=
デフォルト	LINESIZE=オプションのデフォルト値は次の2つのオプションのどちらかによって決定されます。1) キャリッジコントロール文字を含むファイルまたは SAS ログに書き込む場合は、LINESIZE=システムオプションによってデフォルト値が決まります。または 2) ファイルに書き込む場合は、LRECL=オプションによってデフォルト値が決まります。
範囲	最小値は 64 です。最大値は動作環境で許容されている最大論理レコード長です。
操作	LINESIZE=オプションで指定した行の長さを超えるデータの出力を PUT ステートメントが要求した場合の措置は、FLOWOVER オプション、DROPOVER オプション、STOPOVER オプションのうち、どのオプションが指定されているかによって異なります。デフォルト (FLOWOVER) では、データを複数の行に分けて書き出します。
動作環境	LINESIZE=オプションで許容される最大値は、動作環境によって異なります。
注	LINESIZE=オプションには、行のうちデータの出力に使用する長さを指定します。LRECL=オプションには、ファイルの物理的な長さを指定します。
参照項目:	<a href="#">LRECL= (85 ページ)</a> 、 <a href="#">"DROPOVER" (81 ページ)</a> 、 <a href="#">"FLOWOVER" (83 ページ)</a> 、および <a href="#">"STOPOVER" (89 ページ)</a>
例	<a href="#">"例 6: 出力行が出力ファイルの行の長さを超える場合" (95 ページ)</a>

**LINESLEFT=variable**

現在のページの残りの行数を格納する変数の名前を指定します。指定した変数には、現在のページの残りの行数の値が自動的に割り当てられます。LINESLEFT=変数の値は、PUT ステートメントの実行終了時に設定されます。

別名	LL=
ヒント	この変数は、自動変数と同じようにデータセットに書き込まれません。
例	<a href="#">"例 2: 現在のページに残っている行数に基づき新規ページを決定する" (93 ページ)</a>

**LRECL=logical-record-length**

論理レコード長を指定します。この値は、有効なレコード形式によって異なります(RECFM)。LRECL=のデフォルト値は 32,767 です。固定長レコード (RECFM=F) を使用する場合、LRECL=のデフォルト値は 256 になります。最大レコード長は 1G です。

- RECFM=F の場合、LRECL=オプションの値により各出力レコードの長さが決定されます。出力レコードは、切り捨てられるか、指定されたサイズに合うように空白が追加されます。

注: RECFM=F の場合、SAS が SAS9.4 以前のバージョンと通信するには、LRECL=は 256 に設定する必要があります。

- RECFM=N の場合、LRECL=オプションの値は 256 以上にする必要があります。

- RECFM=V の場合、LRECL=オプションの値により最大レコード長が決定されます。指定された長さを超えるレコードは、複数のレコードに分割されます。

**デフォルト** LRECL=オプションの指定を省略すると、動作環境のファイル特性に応じた値が選択されます。

**操作** かわりに、[LRECL システムオプション](#)を使用すると、グローバルな論理レコード長を指定できます。グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。

**注** LINESIZE=オプションには使用するレコードの長さを指定します。LRECL=オプションには出力ファイルの物理的な長さを指定します。

**参照項目:** [LINESIZE= \(84 ページ\)](#)、[PAD \(88 ページ\)](#)、および [PAGESIZE= \(88 ページ\)](#)

## MOD

ファイルに書き込まれたデータをそのファイルに追加する必要があることを示します。

**デフォルト** OLD

**参照項目:** ["OLD" \(87 ページ\)](#)

## NEW | OLD

新規ファイルまたは既存ファイルのどちらを出力に使用するかを指定します。NEW を指定すると、出力用に新規のファイルが開きます。ファイルがすでに存在する場合は、削除された後でファイルが再作成されます。OLD を指定すると、以前のファイルの内容が置き換えられます。

**デフォルト** NEW

## N=*available-lines*

DATA ステップの現在の反復で出力ポインタの移動に使用できる行数を指定します。*available-lines* には、数値(*n*)を指定するか、またはキーワード PAGESIZE や PS を指定できます。

*n*

出力ポインタの移動に使用できる行数を指定します。指定した行数の範囲内で出力中にポインタを前後に動かしてから、ポインタを次の範囲に移動します。

## PAGESIZE

このキーワードを指定すると、出力ポインタをページ全体で移動できます。

**別名** PS

**制限事項** N=PAGESIZE の指定は、出力を表示する場合にのみ有効です。

現在のファイルが出力先のファイルの場合、*available-lines* には、1 または PAGESIZE のどちらかの値を指定する必要があります。

**操作** 出力ポインタの移動に使用できる行数を制御する場合、N=オプション使用するだけでなく、PUT ステートメントで#n 行ポインタコントロールを使用することもできます。

N=オプションの指定を省略し、行ポインタコントロール#を使用しない場合、移動できる範囲は1行です。つまり、デフォルトではN=1に設定されます。N=オプションを使用せず、行ポインタコントロール#を使用する場合、現在のDATAステップのPUTステートメントで使用した行ポインタコントロール#のうち、最大値がN=の値に割り当てられます。

**ヒント** N=PAGESIZE に設定すると、複数の列を含んだページの出力で、列ごとの出力ができます。

**例** [“例 3: ページ全体のコンテンツを配置する” \(93 ページ\)](#)

### ODS <= (ODS-suboptions) >

ODS(Output Delivery System)を使用して DATA ステップの出力をフォーマットします。ここでは、データコンポーネントの構造を定義し、DATA ステップの結果を保持します。また、そのコンポーネントをテーブル定義に関連付け、出力オブジェクトを生成します。ODSはこのオブジェクトを開かれているすべての ODS 出力先に送信し、それぞれの出力先で出力が適切にフォーマットされるようにします。

**デフォルト** ODS-suboptions の指定を省略すると、DATA ステップでは、SASHELP.TMPLMST テンプレートストアに格納されているデフォルトのテーブル定義(base.datastep.table)が使用されます。この定義では2つの標準的な列を定義します。1つは文字変数用、もう1つは数値変数用です。ODSではDATAステップにあるそれぞれの変数を2つの列のどちらかに関連付け、DATAステップで定義された順にその変数を表示します。

ODS-suboptions を指定しない場合、デフォルトのテーブル定義では列のヘッダーに変数のラベルを使用します。ラベルが存在しない場合、デフォルトのテーブル定義では列のヘッダーに変数の名前を使用します。

**制限事項** ODS オプションは、\_FILE\_=オプション、FILEVAR=オプション、HEADER=オプション、PAD オプションと併用できません。

VARCHAR データタイプの変数はサポートされていません。

**要件** ODS オプションは、FILE ステートメントでファイル参照名に PRINT を指定した場合にのみ有効です。

**操作** DELIMITER=オプションおよび DSD オプションは、ODS オプションには適用されません。

### OLD

ファイルの古い内容を置き換えます。

**デフォルト** OLD

**参照項目:** [“MOD” \(86 ページ\)](#)

**PAD | NOPAD**

LRECL=オプションに指定した長さに達するまで、外部ファイルに書き込むレコードに空白を追加するかどうかを指定します。

**デフォルト** 可変長ファイルに書き込む場合は、NOPAD がデフォルト値になります。固定長ファイルに書き込む場合は、PAD がデフォルト値になります。

**ヒント** PAD オプションを指定すると、可変長ファイルの中に固定長レコードを簡単に作成できます。

参照項目: [LRECL= \(85 ページ\)](#)

**PAGESIZE=value**

レポートの 1 ページあたりの行数を設定します。

**別名** PS=

**デフォルト** PAGESIZE=システムオプションの値

**範囲** 15 から 32767 までの値を指定できます。

**操作** TITLE ステートメントが現在定義されている場合、1 ページあたりの行数には、タイトルの行数が含まれます。

**ヒント** PAGESIZE=オプションに指定した値に達すると、出力ポインタは新しいページの 1 行目に移動します。

FILE LOG を指定する場合、最初のページに書き出される行数は、SAS 起動時の注意事項で使用する行数だけ少なくなります。たとえば、PAGESIZE=20 と指定し、SAS 起動時の注意事項に 9 行使用する場合、最初のページに使用できるのは 11 行だけになります。

参照項目: ["PAGESIZE= System Option" \(SAS Viya System Options: Reference\)](#)

**PRINT | NOPRINT**

キャリッジコントロール文字を出力行で使用するかどうかを制御します。

**制限事項** ファイルへ出力する場合、N=オプションの値は 1 または PAGESIZE とする必要があります。

**動作環境** ファイルに出力されるキャリッジコントロール文字は、動作環境によって異なる場合があります。

**ヒント** ファイル参照名に PRINT を使用する場合、PRINT オプションを指定する必要はありません。

対話型 SAS セッションで FILE PRINT を指定する場合、**結果**ウィンドウではフォームフィードコントロール文字をページ区切りと解釈します。また、フォームフィードの前に出力される空行は出力から削除されます。**結果**ウィンドウから結果をフラットファイルに出力すると、改ページ文字を含まないファイルが生成されます。フォームフィード文字をファイルに含める必要がある場合、FILE ステートメントに物理ファイルの場所と PRINT オプションを指定する必要があります。

**RECFM=record-format**

レコード形式を指定します。RECFM は次のいずれかの値を持ちます。

- D            デフォルトのレコード形式(可変長形式)。
- F            固定長形式。つまり、各レコードの長さは同じです。  
RECFM=F は、キャリッジコントロール文字を含む外部ファイルには使用しないでください。
- N            バイナリ形式。ファイルはバイトストリームで構成され、レコード境界はありません。
- P            プリント形式。SAS で、キャリッジコントロール文字が書き込まれます。
- V            可変長形式。各レコードは改行文字で終わります。
- S370V       可変長 S370 レコード形式(V)。
- S370VB      可変長ブロック S370 レコード形式(VB)。
- S370VBS     スパンレコード可変長ブロック S370 レコード形式(VBS)。

**STOPOVER**

PUT ステートメントが現在の行の長さを超えてデータ項目を書き込もうとする場合、DATA ステップの処理を直ちに中止します。その場合、SAS は現在の行の長さを超えたデータ項目を破棄し、エラー発生前に作成された行を書き込んでから、エラーメッセージを発行します。

デフォルト    FLOWOVER

参照項目:    ["FLOWOVER" \(83 ページ\)](#) および ["DROPOVER" \(81 ページ\)](#)

**TERMSTR=**

Linux でフォーマットされたファイルでの行末を制御します。デフォルトでは、ラインフィードのみ、またはキャリッジリターンとラインフィードが行の終わりを示します。行末の区切り文字を明示的に指定するには、次のいずれかの値を指定します。

- CR        キャリッジリターン。
- CRLF     キャリッジリターン、ラインフィード。
- LF        ラインフィード。このパラメータは、Linux によりフォーマットされたファイルの読み込みに使用されます。

**TITLES | NOTITLES**

現在のタイトル行をファイルのページ上に出力するかどうかを制御します。NOTITLES の指定を省略するか、TITLES を指定する場合、定義されているタイトルがあれば出力されます。

別名        TITLE | NOTITLE

デフォルト    TITLES

**UNBUF**

その後の FILE ステートメントでは、バッファされた書き込みをファイルに対して行わないよう SAS に指示します。このオプションは特に、データ収集デバイスに書き込んでいるときに適用されます。

**\_FILE\_=variable**

この FILE ステートメントの現在の出力バッファを参照する文字変数の名前を指定します。この変数は他の変数と同じように使用できます。また、値を

割り当てることもできます。この変数の値は自動的に保持されます。初期値はブランクです。自動変数と同じように、\_FILE\_に指定した変数はデータセットに書き込まれません。

**制限事項** *variable* には、すでに定義済みの変数は指定できません。\_FILE\_=オプションには、この DATA ステップで初めて使用する変数を指定してください。LENGTH ステートメントや ATTRIB ステートメントを使用して、\_FILE\_に指定した変数の長さを設定したり変更することはできません。ただし、ATTRIB ステートメントや FORMAT ステートメントを使用して、この変数に出力形式を指定することができます。

**操作** この文字変数の最大長は、指定した FILE ステートメントの論理レコード長(LRECL)になります。ただし、プログラムの実行直前まで、SAS がこのファイルを開いて LRECL=オプションの値を確認することはありません。そのため、コンパイル実行中は、この変数のサイズは 32,767 バイトになります。

**ヒント** この変数の内容に対する変更は、FILE ステートメントの現在の出力バッファの内容に直ちに反映されます。この FILE ステートメントに対する後続の PUT ステートメントにより、変更されたバッファの内容が書き出されます。N=オプションで複数の出力を指定した場合でも、\_FILE\_に指定した変数は、指定した FILE ステートメントの現在の出力バッファにのみアクセスできます。

他のステートメントで \_FILE\_=オプションを使用せずに出力バッファの内容にアクセスするには、自動変数 \_FILE\_ を使用します。

**参照項目:** [“\\_FILE\\_変数の更新” \(91 ページ\)](#)

## 詳細

### 概要

デフォルトでは、PUT ステートメントの出力は、SAS ログに書き込まれます。FILE ステートメントを使用すると、この出力をプロシジャの出力先と同じ外部ファイル、または別の外部ファイルに書き出すことができます。キャリッジコントロール文字をファイルに追加するかどうかも指定できます。[PRINT | NOPRINT オプション \(88 ページ\)](#)を参照してください。

FILE ステートメントは実行ステートメントなので、条件(IF-THEN)処理で FILE ステートメントを使用できます。1つの DATA ステップでは、複数の FILE ステートメントを使用して複数の外部ファイルに書き込むことができます。

FILE ステートメントでは、DATA ステップの結果の書き込みに ODS(Output Delivery System)を使用できます。

### 外部ファイルの直接更新

FILE ステートメントを INFILE ステートメントや PUT ステートメントと併用すると、レコード全体の更新やレコード内の選択したフィールドのみの更新など、外部ファイルをバッファ内で更新できます。次のガイドラインに従ってください。

- INFILE ステートメントは必ず最初に記述します。
- INFILE ステートメントと FILE ステートメントで、同一のファイル参照名または物理ファイル名を指定します。

- INFILE ステートメントには、FILE ステートメントと INFILE ステートメントの両方で使用できるオプションを使用します。(これらのオプションを FILE ステートメントに指定しても無視されます。)
- INFILE ステートメントに SHAREBUFFERS オプションを指定して、FILE ステートメントと INFILE ステートメントが同一のバッファを使用できるようにします。それによって CPU 時間が節約されるとともに、レコード全体ではなく個々のフィールドを更新することが可能になります。

### 出力バッファのコンテンツへのアクセス

`_FILE_` に指定する変数の他に、`_FILE_` 自動変数を使用しても最後に実行した FILE ステートメントの出力バッファの内容を参照できます。この文字変数の値は自動的に保持されます。初期値はブランクです。他の自動変数と同じように、`_FILE_` 自動変数はデータセットに書き込まれません。

INFILE ステートメントに `_FILE_` オプションを指定すると、指定した変数の値は自動変数 `_FILE_` でも間接的に参照されます。自動変数 `_FILE_` を指定し、特定の FILE ステートメントで `_FILE_` オプションの指定を省略した場合、`_FILE_` オプションの変数とその FILE ステートメントに対して内部的に作成されます。これ以外の場合に、特定の FILE ステートメントに対して `_FILE_` オプションの変数が作成されることはありません。

実行時または参照時、自動変数 `_FILE_` の最大長は、そのときに使用している `_FILE_` オプションの変数の最大長に一致します。ただし、自動変数 `_FILE_` が参照している他の変数の長さは実行時まで確定されないため、自動変数 `_FILE_` の長さはコンパイル時に 32,767 バイトに設定されます。たとえば、自動変数 `_FILE_` の値を長さが定義されていない新しい変数に割り当てる場合、新しい変数のデフォルトの長さは 32,767 バイトになります。LENGTH ステートメントや ATTRIB ステートメントを使用して、自動変数 `_FILE_` の変数の長さを設定したり、無効にすることはできません。ただし、FORMAT ステートメントや ATTRIB ステートメントを使用すると、自動変数 `_FILE_` に出力形式を割り当てることができます。

### `_FILE_` 変数の更新

他の SAS 変数と同じように、自動変数 `_FILE_` を更新することができます。次の 2 つの方法を使用できます。

- 割り当てステートメントで自動変数 `_FILE_` を使用する
- PUT ステートメントを使用する

次の形式で割り当てステートメントを使用すると、`_FILE_` 変数を更新できます。

```
_FILE_ = <'string-in-quotation-marks' | character-expression>
```

この割り当てステートメントでは、現在の出力バッファの内容を更新し、バッファの長さを `'string-in-quotation-marks'` または `character-expression` の長さに設定します。ただし、割り当てステートメントを使用しても、PUT ステートメントの現在の出力ポインタの位置は変化しません。この FILE ステートメントの次の PUT ステートメントに対しては列 1 から、PUT ステートメントに後置 @ を使用する場合は既知の最終位置からバッファの更新を開始します。

次の例では、割り当てステートメントで現在の出力バッファの内容を更新します。PUT ステートメントのカラムポインタの位置は変わりません。

```
file print;
_file_ = '_FILE_';
put 'This is PUT';
```

次の出力が作成されます。This is PUT

この例では、



```
file print;
_file_ = 'This is from FILE, sir.';
put @14 'both';
```

次の出力が作成されます。 **This is from both, sir.**

PUT ステートメントを使用しても、自動変数\_FILE\_を更新することができます。PUT ステートメントでは出力バッファのデータに出力形式を設定し、自動変数\_FILE\_がそのバッファを参照するため、自動変数\_FILE\_は PUT ステートメントで更新されます。ただし、デフォルトでは、この出力バッファは、PUT ステートメントを実行し、現在のレコード(または、N=オプションに指定したレコードブロック)を出力した後に消去されます。そのため、出力前に自動変数\_FILE\_の内容の確認や変更を行うには、PUT ステートメントに後置@または後置@@を指定します(N=1 の場合)。値が N=1 以外の場合、行ポインタコントロールの最終位置をレコードブロックの最後のレコードに指定する PUT ステートメントで、後置@または後置@@を使用します。次の例は、N=1 の場合です。

```
file ABC;
put 'Something' @;
Y = _file_||' is here';
file ABC;
put 'Nothing' ;
Y = _file_||' is here';
```

Y には最初に **Something is here** が割り当てられ、次に **is here** が割り当てられます。

自動変数\_FILE\_に変更を加えると、現在の FILE ステートメントの出力バッファに直ちに変更が反映されます。FILE ステートメントに続けて PUT ステートメントを実行すると、変更したバッファの内容が出力されます。

N=オプションに複数のバッファを指定した場合でも、自動変数\_FILE\_は、その FILE ステートメントで現在使用中の出力バッファにのみアクセスします。N=オプションに指定したすべてのバッファにアクセスできますが、特定のバッファを現在の出力バッファに設定するには、PUT ステートメントに#行ポインタコントロールを指定する必要があります。

## 比較

- FILE ステートメントは、PUT ステートメントの出力ファイルを指定します。INFILE ステートメントは、INPUT ステートメントの入力ファイルを指定します。
- FILE ステートメントおよび INFILE ステートメントでは、使用する外部ファイルに関する追加情報を SAS システムに提供するオプションを使用できません。

## 例

### 例 1: 新規ページの開始時にステートメントを実行する

次の DATA ステップは、HEADER=オプションの使用法を示しています。

- レポートを出力しません。DATA\_NULL\_を使用して、データセットの作成ではなく、レポートの出力を行うように指示します。

```
data _null_;
set sprint;
by dept;
```



- **結果ウィンドウに出力を送信します。ヘッダー情報を参照します。** ファイル参照名に PRINT を指定し、プロシジャの出力先と同じ場所へ出力を送信します。HEADER=オプションは、各ページのヘッダーを作成するステートメントの前に配置されているラベルを参照します。

```
file print header=newpage;
```

- **部門ごとに新しいページを開始します。**

```
if first.dept then put _page_;
put @22 salesrep @34 salesamt;
```

- **各ページにヘッダーを出力します。新しいページを開始するたびに、次のステートメントが実行されます。** ラベルが指定されたグループの最後のステートメントとして、ラベルの前に RETURN ステートメントを指定する必要があります。

```
return;
newpage:
put @20 'Sales for 1989' /
    @20 dept=;
return;
run;
```

## 例 2: 現在のページに残っている行数に基づき新規ページを決定する

次の DATA ステップでは、LINESLEFT=オプションを使用し、現在のページに残っている行数に基づいて改ページを実行する位置を決定する例を示します。

- **レポートを出力します。** DATA\_NULL\_ を使用して、データセットの作成ではなく、レポートの出力を行うように指示します。

```
data _null_;
set info;
```

- **標準の結果ウィンドウに出力を送信します。** ファイル参照名に PRINT を指定し、プロシジャの出力先と同じ場所へ出力を送信します。LINESLEFT オプションは、変数 REMAIN に現在のページに残っている行数が格納されていることを示しています。

```
file print linesleft=remain pagesize=20;
put @5 name @30 phone
    @35 bldg @37 room;
```

- **現在のページの残りの行数が指定した値よりも少なくなると、新しいページが開始されます。** この条件に従って、PUT\_PAGE\_ は新しいページを開始し、ポインタを 1 行目に配置します。

```
if remain<7 then put _page_;
run;
```

## 例 3: ページ全体のコンテンツを配置する

この例では DATA ステップで N=PAGESIZE オプションを使用し、2 列配置された電話番号リストを生成する方法を示しています。1 つの列には名前を格納し、もう 1 つの列には電話番号を格納します。

- **レポートを作成し、結果ウィンドウに書き込みます。** DATA\_NULL\_ を使用して、データセットの作成ではなく、レポートの出力を行うように指示します。ファイル参照名は PRINT です。キャリッジコントロール文字を使用して、出力するファイルの特性にあった出力を行います。N=PAGESIZE と指定すると、出力ポインタをページ全体に移動できます。

```

data _null_;
  file 'external-file' print n=pagesize;

```

- レポートに使用する列を指定します。DATA ステップの繰り返しごとに、DO ループを 2 回繰り返します。COL の値は最初の繰り返しで 1 に設定され、2 回目の繰り返しで 40 に設定されます。

```

do col=1, 40;

```

- データを 20 行書き込みます。次の DO ループを 20 回繰り返して列 1 に 20 行データを書き込みます。書き込みが完了すると、外側のループで COL が 40 に設定されます。この DO ループを 20 回繰り返し、列 2 にデータを 20 行書き込みます。LINE および COL の値は DO ステートメントによって設定され、増分が追加されます。この LINE と COL の値によって、PUT ステートメントでページ上に NAME と PHONE の値を書き込む位置が制御されます。

```

do line=1 to 20;
  set info;
  put #line @col name $20. +1 phone 4.;
end;

```

- 2 列分のデータがすべて揃ったら、ページ全体を出力します。次の END ステートメントで外側のループを終了します。PUT \_PAGE\_ は現在のページを出力し、ポインタを次のページの最上部に移動させます。

```

end;
  put _page_;
run;

```

#### 例 4: 現在の出力ファイルの識別

次の DATA ステップでは、ファイル識別メッセージをログに出力し、現在の出力ファイルの値を変数 MYOUT に割り当てます。次の PUT ステートメントは、MYOUT に適切な値を割り当ててから、この変数の値を出力ファイルに書き込む例を示しています。

```

data _null_;
  length myout $ 200;
  file file-specification filename=myout;
  put myout=;
  stop;
run;

```

次の PUT ステートメントは、現在の出力ファイルに、ファイルの物理名が記述された行を書き込みます。

```
MYOUT=your-output-file
```

#### 例 5: 現在の出力ファイルを動的に変更する

次の DATA ステップでは、FILEVAR=オプションを使用して、現在開かれている出力ファイルを新しい物理ファイルに動的に変更します。

- レポートを出力します。その次に、長い文字変数を作成します。DATA \_NULL\_ を使用して、データセットの作成ではなく、レポートの出力を行うように指示します。LENGTH ステートメントを使用して、外部ファイル名を格納するのに十分な長さを割り当てた変数を作成します。

```

data _null_;
  length name $ 200;

```

- インストリームデータ行を読み込み、NAME 変数に値を割り当てます。

```
input name $;
```

- *NAME* 変数の値が変わったら、現在の出力ファイルを閉じ、新しいファイルを開きます。*file-specification* は、プレースホルダです。これには、有効な SAS 名を指定できます。

```
file file-specification filevar=name mod;
date = date();
```

- 現在開かれている出力ファイルにログレコードを追加します。

```
put 'records updated ' date date.;
```

- 外部ファイルの名前を指定します。

```
datalines;
external-file-1
external-file-2
external-file-3
;
```

### 例 6: 出力行が出力ファイルの行の長さを超える場合

変数の行の長さを合計すると出力行(80 文字)よりも長くなるので、次の PUT ステートメントでは、自動的に 3 つのレコードに分割して書き込みます。

```
file file-specification linesize=80;
put name $ 1-50 city $ 71-90 state $ 91-104;
```

変数 *NAME* の値は 1 番目のレコードに出力されます。変数 *CITY* の値の出力は 2 番目のレコードの列 1 から、変数 *STATE* の値の出力は 3 番目のレコードの列 1 から始まります。

### 例 7: 出力ファイル書き込み時のエンコードの指定

この例では、SAS データセットから外部ファイルを作成します。現在のセッションエンコーディングは *Wlatin1* ですが、外部ファイルのエンコーディングは UTF-8 にする必要があります。デフォルトでは、外部ファイルは現在のセッションエンコーディングを使用して書き込まれます。

外部ファイルへの書き込み時に使用するエンコーディングを指定するには、*ENCODING=* オプションを指定します。外部ファイルを UTF-8 エンコーディングにするように指示すると、外部ファイルへの書き込み時に、*Wlatin1* から指定した UTF-8 エンコーディングにデータがトランスコードされます。

```
libname myfiles 'SAS-library';
filename outfile 'external-file';
data _null_;
set myfiles.cars;
file outfile encoding="utf-8";
put Make Model Year;
run;
```

### 例 8: SFTP アクセス方式を使用して Excel スプレッドシートにデータを出力する

この例では、FTP アクセス方式と *FILEVAR* オプションを使用して、複数の Microsoft Excel ワークシートにデータを書き込みます。

```
data _null_;
do i = 1 to 3;
sheet = cats('excel|[test-sheet.xlsx]Sheet', i, '!r1c1:r10c2');
file area sftp filevar=sheet;
```

```

do x = 1 to 10;
  y = 2*x;
  put x y;
end;
end;
run;

```

## 関連項目:

### ステートメント:

- [“FILENAME ステートメント” \(96 ページ\)](#)
- [“INFILE ステートメント” \(146 ページ\)](#)
- [“LABEL ステートメント” \(217 ページ\)](#)
- [“LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide](#)
- [“PUT ステートメント” \(318 ページ\)](#)
- [“RETURN ステートメント” \(365 ページ\)](#)
- [“TITLE ステートメント” \(399 ページ\)](#)

### システムオプション:

- [“LOCKDOWN System Option in SAS Intelligence Platform: Application Server Administration Guide](#)

---

## FILENAME ステートメント

SAS ファイル参照名を外部ファイルまたは出力デバイスと関連付けたり、ファイル参照名と外部ファイルの関連付けを取り消します。また、外部ファイルの属性を書き込みます。

**該当要素:** 任意の場所

**カテゴリ:** データアクセス

**制限事項:** このステートメントは CAS Server ではサポートされていません。  
SAS がロックダウン状態にある場合、ロックダウンパスリストに含まれていないファイルに関しては、FILENAME ステートメントを使用できません。

---

## 構文

形式 1: **FILENAME** *fileref* <*device-type*> '*external-file*' <ENCODING=*encoding-value*> <*options*> <*Linux-options*>;

形式 2: **FILENAME** *fileref* *device-type* <*options*> <*Linux-options*>;

形式 3: **FILENAME** *fileref* CLEAR | \_ALL\_ CLEAR;

形式 4: **FILENAME** *fileref* LIST | \_ALL\_ LIST ;

形式 5: **FILENAME** *fileref* ('*pathname-1*' ... '*pathname-n*') <ENCODING=*encoding-value*> <'*Linux-options*'> <LOCKINTERNAL= AUTO | SHARED>;

形式 6: **FILENAME** *fileref* *directory-name* <ENCODING=*encoding-value*>

<LOCKINTERNAL= AUTO | SHARED>;

## 引数

### *fileref*

新しいファイル参照名を割り当てるときに使用する SAS 名を指定します。現在割り当てられているファイル参照名の関連付けを取り消したり、FILENAME ステートメントを使用してファイルの属性をリストする場合は、FILENAME ステートメントに割り当て済みのファイル参照名を指定するか動作環境レベルのコマンドを指定します。

ヒント ファイル参照名と外部ファイルの関連付けは、SAS セッション終了まで維持されるか、または他の FILENAME ステートメントで関連付けの変更や関連付けの取り消しを実行するまで維持されます。ファイルに対するファイル参照名は必要に応じて何度でも変更できます。

### *device-type*

ファイル参照名が入力デバイスまたは出力デバイスを参照する場合や物理ファイルではない場所を参照する場合に使用する、デバイスタイプまたはアクセス方式を指定します。

#### ACTIVEMQ

このアクセス方式を指定すると、ActiveMQ メッセージブローカーへアクセスできます。

操作 DATA ステップでアクセス方式のオプションを認識できない場合、DATA ステップはこのオプションをアクセス方式に渡して処理しません。

#### DISK

デバイスがディスクドライブであると指定します。

ヒント ディスク上のファイルにファイル参照名を割り当てる場合、DISK を指定する必要はありません。

#### DUMMY

ファイルへの出力を破棄するように指定します。

ヒント テストを実行する場合は DUMMY を指定すると便利です。

#### GTERM

出力デバイスの種類がグラフィックデータを受信するグラフィックデバイスであると指定します。

#### JMS

Java Message Service (JMS) の送信先を指定します。

#### PIPE

名前の付いていないパイプを指定します。

注 動作環境によっては、パイプがサポートされない場合があります。

#### PLOTTER

バッファなしのグラフィック出力デバイスを指定します。

#### PRINTER

プリンタまたはプリンタスプールファイルを指定します。

**SFTP**

このアクセス方式を指定すると、SFTP プロトコルを使用してリモートファイルにアクセスできるようになります。

参照項目: [“FILENAME ステートメント、SFTP アクセス方式” \(109 ページ\)](#)

**TAPE**

テープドライブを指定します。

**TEMP**

ファイル名が割り当てられている間だけ存在する一時ファイルを作成します。この一時ファイルは論理名からのみアクセスできます。また、論理名が存在する間だけ使用できます。

**制限事項** 物理パス名は指定しないでください。物理パス名を指定するとエラーが発生します。

**ヒント** TEMP デバイスで操作するファイルは、DISK ファイルに対して、同じ属性を保有し、同じように動作します。

**TERMINAL**

ユーザーの端末を指定します。

**UPRINTER**

ユニバーサル印刷プリンタの定義名を指定します。

**ヒント** FILENAME ステートメントにプリンタ名を指定しない場合、PRINTERPATH オプションによって、使用するユニバーサルプリンタと出力先が制御されます。

**URL**

このアクセス方式を指定すると、URL アクセス方式を使用してリモートファイルにアクセスできるようになります。

参照項目: [“FILENAME ステートメント、URL アクセス方式” \(115 ページ\)](#)

**ZIP**

このアクセス方式を指定すると、ZIP ファイルにアクセスできるようになります。

参照項目: [“FILENAME ステートメント、ZIP アクセス方式” \(121 ページ\)](#)

**要件** ステートメント中、*device-type* の直後に *fileref* が必要です。

参照項目: [“FILENAME ステートメントのデバイス情報” \(104 ページ\)](#)

**'external-file'**

外部ファイルの物理名です。物理名には動作環境で判別できる名前を指定します。

**ヒント** 外部ファイルにファイル参照名を割り当てる場合は、*external-file* を指定します。

ファイル参照名には、1 つのファイルまたは集約記憶域を関連付けることができます。

参照項目: [“Specifying Pathnames in Linux” \(Batch and Line Mode Processing in SAS Viya\)](#)

---

“FILENAME ステートメントのデバイス情報” (104 ページ)

---

**'pathname-1' ... 'pathname-n'**

ユーザーが同一のファイル参照名を使用してアクセスしようとしているファイルのパス名です。ファイル名を連結したいときは、この形の FILENAME ステートメントを使用します。ファイル名の連結は DISK ファイルの場合のみ可能です。したがって、*device-type* を指定する必要はありません。カンマまたは空白スペースを使ってパス名を区切ります。それぞれのパス名を引用符で囲みます。“[Specifying Pathnames in Linux Environments](#)” (*Batch and Line Mode Processing in SAS Viya*) に、パス名の指定に使える文字置換機能を示します。If the fileref that you are defining is to be used for input, then you can also use wildcards. Remember that Linux ファイル名には大文字と小文字の区別があります。

**directory-name**

アクセスするファイルが含まれるディレクトリを指定します。詳細については、“[Assigning a Fileref to a Directory \(Using Aggregate Syntax\)](#)” (*Batch and Line Mode Processing in SAS Viya*) を参照してください。

**ENCODING= 'encoding-value'**

外部ファイルからの読み込みや外部ファイルへの書き込みに使用するエンコーディングを指定します。ENCODING=の値は、外部ファイルのエンコーディングが現在のセッションエンコーディングとは異なることを示しています。

外部ファイルからデータを読み込む場合は、指定したエンコーディングからセッションエンコーディングにデータがトランスコードされます。外部ファイルにデータを書き込む場合は、セッションエンコーディングから指定したエンコーディングにデータがトランスコードされます。

**デフォルト** SAS では、外部ファイルのエンコーディングがセッションエンコーディングと同じであるとみなします。

**制限事項** UPRINTER デバイスタイプは ENCODING=引数をサポートしません。

---

FILENAME ステートメントで、PROC CPORT で生成した移送ファイルのエンコーディングを指定することはできません。移送ファイルをインポートするためには、移送元と移送先の SAS セッションのエンコーディングに互換性がなければなりません。

**参照項目:** エンコーディングの有効な値については、“[Encoding Values in SAS Language Elements](#)” (*SAS Viya National Language Support: Reference Guide*) を参照してください。

**例** “例 5: 外部ファイル読み込み時のエンコードの指定” (108 ページ)

“例 6: 外部ファイル書き込み時のエンコードの指定” (108 ページ)

---

**LOCKINTERNAL=AUTO | SHARED**

FILENAME ステートメントにリスト表示されているファイルに使用する、SAS System のロックを指定します。LOCKINTERNAL は次のいずれかの値を持ちます。

**AUTO**

SAS セッションで、あるユーザーがあるファイルへの書き込みアクセスを持っている場合、他のユーザーはそのファイルへの読み込みまたは書き込みアクセスを持てないように、ファイルをロックします。あるユーザーが

あるファイルへの読み込みアクセスを持っている場合、他のユーザーはそのファイルへの書き込みアクセスを持ってませんが、複数のユーザーが読み込みアクセスを持つことは可能です。

### SHARED

SAS セッションで、2人のユーザーが同時に同じファイルへのアクセスを持ってないように、ファイルをロックします。その場合ファイルは、書き込みアクセスを持っている1人のユーザーと読み込みアクセスを持っている複数のユーザーが、同時に共有できます。

デフォルト AUTO

### CLEAR

現在割り当てられている1つまたは複数のファイル参照名の関連付けを取り消します。

**動作** 環境変数によって定義されるファイル参照名を削除することはできません。環境変数によって定義されるファイル参照名は、SAS セッション全体に割り当てられます。

**ヒント** 1つのファイル参照名の関連付けを取り消すには、*fileref* を指定します。現在割り当てられているファイル参照名の関連付けをすべて取り消すには、*\_ALL\_* を指定します。

### *\_ALL\_*

現在割り当てられているすべてのファイル参照名に対して、CLEAR 引数または LIST 引数を適用するように指定します。

### LIST

1つまたは複数のファイルの属性を SAS ログに書き込みます。

**操作** 1つのファイルの属性を書き込むには、*fileref* を指定します。現在のセッションにあるファイル参照名を含むすべてのファイルの属性を書き込むには、*\_ALL\_* を指定します。

**動作環境** 環境変数として定義されているファイル参照名は、それらがすでに SAS ステートメントで使用されている場合のみ、表示されます。Bourne シェルや Korn シェルを使っている場合、SAS は事前に開いたファイルの名前を判断できないので、ファイル名の代わりに「<File Descriptor number>」という形の文字列を表示します。

## オプション

### RECFM=*record-format*

レコード形式を指定します。RECFM は次のいずれかの値を持ちます。

- D デフォルトのレコード形式(可変長形式)。
- F 固定長形式。つまり、各レコードの長さは同じです。外部ファイルがキャリッジ制御文字でレコードを区切る形式の場合、RECFM=F は指定しないでください。
- N バイナリ形式。ファイルはバイトストリームで構成され、レコード境界はありません。N は、PIPE デバイスタイプでは無効です。LRECL オプションを指定しない場合、デフォルトではファイルから 256 バイトが読み込まれます。
- P プリント形式。SAS は出力時にキャリッジ制御文字を書き出します。



V	可変長形式。各レコードは改行文字で終わります。
S370V	可変長 S370 レコード形式(V)。
S370VB	可変長ブロック S370 レコード形式(VB)。
S370VBS	スパンレコード可変長ブロック S370 レコード形式(VBS)。 RECFM=S370VBS を指定する場合は、255 文字を超えるレコードに発生するエラーを回避するために BLKSIZE=32,760 および LRECL=32,760 を指定する必要があります。

操作 グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。

## Linux のオプション

### 'Linux-options'

は Linux 環境に特有です。これらのオプションでは次のどの値でも使用できます。

#### BLKSIZE=

#### BLK=

1 回の I/O 操作で物理的に書き込まれるまたは読み込まれるバイト数を指定します。デフォルト値は 64K です。最大値は 1G-1 です。

RECFM=S370VBS を指定する場合は、255 文字を超えるレコードに発生するエラーを回避するために BLKSIZE=32,760 を指定する必要があります。

#### LRECL=

論理レコード長を指定します。この値は、有効なレコード形式によって異なります(RECFM)。LRECL=のデフォルト値は 32,767 です。固定長レコード(RECFM=F)を使用する場合、LRECL=のデフォルト値は 256 になります。最大長は 1G です。

- RECFM=F の場合、LRECL=オプションの値は、1 件のレコードとして読み込まれるバイト数または各出力レコードの長さを決定します。出力レコードは、切り捨てられるか、指定されたサイズに合うように空白が追加されます。

注: RECFM=F の場合、SAS の以前のバージョンでのデフォルト値を使用して作成された固定長レコードを読み込むときには、LRECL=を 256 に設定する必要があります。

- RECFM=N の場合、LRECL=オプションの値は 256 以上にする必要があります。
- RECFM=V の場合、LRECL=オプションの値により最大レコード長が決定されます。指定された長さを超えるレコードは、出力上では複数のレコードに分割され、入力上では切り捨てられます。
- RECFM=S370VBS を指定する場合は、255 文字を超えるレコードに発生するエラーを回避するために LRECL=32,760 を指定する必要があります。

#### MOD

ファイルに書き込まれたデータをそのファイルに追加する必要があることを示します。

#### NEW | OLD

新規ファイルまたは既存ファイルのどちらを出力に使用するかを指定します。NEW を指定すると、出力用に新規のファイルが開きます。ファイ

ルがすでに存在する場合は、削除された後でファイルが再作成されます。OLD を指定すると、以前のファイルの内容が置き換えられます。NEW がデフォルト値です。

#### PERMISSION='permission-value'

指定ファイル参照名に対して設定する権限を指定します。2つ以上の権限値セットを指定するには、引用符内でカンマを使用して区切ります。

次の形式で *permission-value* を指定します。

#### A::<trustee\_type>::<permissions>

'A'は、これらがアクセス権限であることを示します。現在サポートされている値は他にありません。

trustee\_type が取り得る値を次に示します。

- u ユーザー
- g グループ(ファイルのグループ所有者)
- o その他(他のユーザーすべて)

権限値は、英文字 *r* (読み取り)、*w* (書き込み)、および *x* (実行)の値をこの順序で取ります。これらの権限の1つを付与しない場合は、その位置に '-' を入力します(たとえば、*r-x* や *rw-*など)。

ファイル参照名に対して読み取り、書き込み、および実行権限を得たいとします。また、ファイルのグループ所有者に対して読み取りおよび実行権限を指定するとします。最後に、他のユーザーすべてがファイルに対して読み取り権限のみを得ることを許可します。これらのオプションは、次のように指定できます。

```
permission='A::u::rwx,A::g::r-x,A::o::r-'
```

3つのトラスティの種類すべてに対して権限値を指定します。権限値のリストで省略したトラスティの種類はいずれも、指定ファイル参照名へのアクセスをすべて拒否されます。たとえば、次の権限値を使用したとします。

```
permission='A::u::rwx,A::g::r-x'
```

この場合、指定ファイルにアクセスできるのは所有者およびグループ所有者のみになります。所有者およびグループ所有者以外のユーザーはいずれもファイルへのアクセスをすべて拒否されます。

#### TERMSTR=

Linux でフォーマットされたファイルでの行末を制御します。デフォルトでは、ラインフィード単独でも、キャリッジリターンとラインフィードが重なっていても、行末と判断します。行末の区切り文字を明示的に定義したい場合は、次のいずれかを指定してください。

- CR キャリッジリターン。
- CRLF キャリッジリターン、ラインフィード。PC用にフォーマットされたファイルを読み込む場合は、この値を使用します。CRLF がデフォルトです。
- LF ラインフィード。

Linux 上で読み込むことになるファイルを書き込む場合は、TERMSTR=LF を指定してください。

#### UNBUF

その後の FILE ステートメントでは、バッファされた書き込みをファイルに対して行わないよう SAS に指示します。このオプションは、特にユー

ザーがデータ収集デバイスに対して読み込みまたは書き込みを行っているときに適用されます。SAS Viya ステートメント: リファレンスで説明したように、INFILE ステートメントでもバッファされた読み込みを行わないようになります。

## 詳細

### 定義

#### 外部ファイル

動作環境に作成および保存されているファイルです。このファイルは、データ、SAS プログラミングステートメント、自動呼び出しマクロの読み込み先または出力の書き込み先となります。外部ファイルには、1つのファイルまたは複数の外部ファイルを格納する集約記憶域を指定できます。“例 3: ファイル参照名を集約記憶域に関連付ける” (107 ページ)を参照してください。

#### fileref

ファイル参照名とは、外部ファイルの参照先を簡略化したものです。外部ファイルにファイル参照名を関連付けると、SAS プログラミングステートメント(INFILE、FILE、および%INCLUDE など)や、外部ファイルにアクセスする SAS ソフトウェアの他のコマンドやステートメントで簡略化した参照先としてファイル参照名を使用できます。

### 外部ファイルからの区切られたデータの読み込み

テキストファイルがローカルのエンコーディング環境以外の場所で作成された場合、EBCDIC 環境または ASCII 環境のどちらかで ENCODING=オプションを指定する必要があります。

たとえば、ASCII プラットフォーム上で EBCDIC ファイルを読み込む場合、FILENAME ステートメントに ENCODING=オプションを指定することをお勧めします。ただし、FILENAME ステートメントで DSD オプションおよび DLM オプションを使用する場合は、ENCODING=オプションを必ず指定する必要があります。これは、2つのオプションがセッションエンコーディングの特定の文字(引用符、カンマ、ブランクなど)を必要とするためです。

エンコーディング固有の入力形式は、バイナリファイルに対して専用で使用する必要があります。このバイナリファイルには、文字フィールドと文字以外のフィールドが含まれます。

### 外部ファイルにファイル参照名を関連付ける(形式 1)

ディスク上の外部ファイルにファイル参照名を関連付けるには FILENAME ステートメントを次の形式で使用します。

```
FILENAME fileref 'external-file' <operating-environment-options>;
```

ディスクファイル以外のファイルにファイル参照名を関連付ける場合、動作環境によっては、次の形式に示すようにデバイスタイプを指定する必要があります。

```
FILENAME fileref <device-type> <operating-environment-options>;
```

ファイル参照名と外部ファイルの関連付けは、SAS セッション終了まで維持されるか、または他の FILENAME ステートメントで関連付けの変更や関連付けの取り消しを実行するまで維持されます。ファイルに対するファイル参照名は必要に応じて何度でも変更できます。

文字セットのエンコーディングを指定するには、次の形式を使用します。

```
FILENAME fileref <device-type> <operating-environment-options>;
```

## 端末、プリンタ、ユニバーサルプリンタ、プロッタにファイル参照名を関連付ける(形式2)

出力デバイスにファイル参照名を関連付ける形式は次のとおりです。

```
FILENAME fileref device-type <operating-environment-options>;
```

## 外部ファイルからファイル参照名の関連付けを取り消す(形式3)

ファイルからファイル参照名の関連付けを取り消すには、ファイル参照名と CLEAR オプションを指定して FILENAME ステートメントを使用します。

```
FILENAME fileref CLEAR | _ALL_ CLEAR;
```

## ファイル属性を SAS ログに書き込む(形式4)

1 つまたは複数の外部ファイルの属性を SAS ログに書き込むには、FILENAME ステートメントを使用します。fileref を指定すると、1 つのファイルの属性を書き込みます。\_ALL\_ を使用すると、現在の SAS セッションにある割り当て済みのファイル参照名を含むすべてのファイルの属性をすべて書き込みます。

```
FILENAME fileref LIST | _ALL_ LIST;
```

## FILENAME ステートメントのデバイス情報

次の表の、デバイスタイプまたはアクセス方法と、関連する外部ファイルとの関係を示します。

表 2.3 FILENAME ステートメントのデバイス情報

デバイスまたはアクセス方法	関数	外部ファイル
ACTIVEMQ	SAS プログラムで、HTTP プロトコルによる ActiveMQ メッセージブローカとのメッセージの送受信を可能にします。	ActiveMQ RESTful API を使用した URL を介してアクセスされます。ActiveMQ MessageServlet が、HTTP 要求と ActiveMQ メッセージディスパッチャ間を統合しています。
DISK	ファイル参照名と DISK ファイルを関連付けます。	単独ファイルのパス名です。ファイル名を連結する場合は、カンマまたはスペースで区切られカッコで囲まれたパス名のリストです。仕様のレベルは、ユーザーのファイルシステム内の場所によって異なります。 <a href="#">"Specifying Pathnames in Linux" (Batch and Line Mode Processing in SAS Viya)</a> に、UNIX のパス名を指定する際に使える文字置換機能を示します。
DUMMY	ファイル参照名をヌルデバイスと関連付けます。	なし。DUMMY を使用すると、デバイスでの読み込み/書き込みを行うことなくアプリケーションをデバッグできます。このデバイスへの出力は破棄されます。
JMS	SAS プログラムで、JMS API 準拠メッセージサービスとのメッセージの送受信を可能にします。	JMS API 仕様を実装している、サードパーティのメッセージ指向ミドルウェアベンダによる JMS プロバイダを介してアクセスされます。classpath に指定が必要です。

デバイスまたはアクセス方法	関数	外部ファイル
PIPE	Linux コマンドから入力を読み込み、あるいはこのコマンドに出力を書き込みます。	Linux コマンドです。詳細については、 <a href="#">“Printing and Routing Output” (SAS Viya Universal Printing)</a> を参照してください。
PLOTTER	出力をプロッタに送信します。	デバイス名およびプロッタオプションです。詳細については、 <a href="#">“Using the PRINTTO Procedure in LINUX Environments” (SAS Viya Universal Printing)</a> を参照してください。 <a href="#">“Assigning Filerefs to External Files or Devices with the FILENAME Statement” (Batch and Line Mode Processing in SAS Viya)</a> .
PRINTER	出力をプリンタに送信します。	デバイス名およびプリンタオプションです。詳細については、 <a href="#">“Printing and Routing Output” (SAS Viya Universal Printing)</a> を参照してください。
SFTP	SSH サーバーが稼働しているネットワーク上のホストコンピュータとの間でファイルを読み書きします。	リモートコンピュータ上の外部ファイルのパス名で、後ろに SFTP オプションが付きます。詳細については、 <a href="#">“FILENAME ステートメント、SFTP アクセス方式” (109 ページ)</a> を参照してください。
TEMP	ファイル参照名を、Work ライブラリに保存されている外部ファイルと関連付けます。	なし
TERMINAL	ファイル参照名をターミナルと関連付けます。	ターミナルのパス名です。
UPRINTER	<b>Printer Setup</b> ダイアログボックスを使ってセットアップされたデフォルトのプリンタに、出力を送信します。	なし
URL*	ファイルの URL を使用してファイルにリモートアクセスできるようにします。	URL サーバー上で読み込みまたは書き込みを行いたいファイルの名前です。URL は次のいずれかの形式で指定する必要があります。 <code>http://hostname/file</code> <code>http://hostname:portno/file</code>

デバイスまたはアクセス方法	関数	外部ファイル
ZIP	ZIP ファイルにアクセスできるようにします。	ZIP ファイルのパス名です。ZIP zlib サービスを利用して、zlib に対応した Linux マシンとの間で読み書きできます。詳細については、“ <a href="#">FILENAME ステートメント、ZIP アクセス方式</a> ” (121 ページ)を参照してください。

\* SAS がロックダウン状態の場合、アクセス方法は、システム管理者が再有効化しない限りアクセス不能(無効)です。

## ファイルロック

外部ファイルのファイルロックは、LOCKINTERNAL オプションによって FILENAME ステートメントレベルで制御されます。LOCKINTERNAL として AUTO (デフォルト値) を指定すると、SAS はファイルを排他的にロックして、書き込みアクセスを持つ 1 人のユーザーしかアクセスできないようにします。SAS はファイルを非排他的にロックして、読み込みアクセスを持つ複数のユーザーがアクセスできるようにします。たとえば、UPDATE または OUTPUT モードでファイルが開かれた場合、内部プロセスからのその他のアクセスすべてがブロックされます。ファイルが INPUT モードで開かれた場合、複数のユーザーがそのファイルを読み込むことができますが、UPDATE 機能および OUTPUT 機能はブロックされます。

LOCKINTERNAL に SHARED 値を使用する場合は、1 人のユーザーがファイルの書き込みアクセスを許可され、複数のユーザーがそのファイルの読み込みを許可されます。

## 比較

FILENAME ステートメントでは、ファイル参照名が外部ファイルに割り当てられます。LIBNAME ステートメントは、SAS データライブラリにライブラリ参照名を割り当てます。DBMS テーブルにアクセスするには、LIBNAME、SAS/ACCESS ステートメントを使用します。

## 例

### 例 1: ファイル参照名または物理的なファイル名の指定

外部ファイルは、外部ファイルにファイル参照名を割り当ててからその参照名を指定するか、物理ファイル名を一重引用符で囲んで指定します。

```
filename sales 'your-input-file';
data jansales;
  /* specifying a fileref */
  infile sales;
  input salesrep $20. +6 jansales febsales marsales;
run;
data jansales;
  /* physical filename in quotation marks */
  infile 'your-input-file';
  input salesrep $20. +6 jansales febsales marsales;
run;
```

**例 2: FILENAME ステートメントと LIBNAME ステートメントの使用**

この例では、ファイル参照名 GREEN が関連付けられているファイルからデータを読み込み、永久 SAS データセットを作成します。このデータセットは、ライブラリ参照名 SAVE が関連付けられている SAS ライブラリに格納されます。

```
filename green 'your-input-file';
libname save 'SAS-library';
data save.vegetable;
  infile green;
  input lettuce cabbage broccoli;
run;
```

**例 3: ファイル参照名を集約記憶域に関連付ける**

ファイル参照名を集約記憶域に関連付ける場合、ファイル参照名を使用し、その後集約記憶域に格納されている読み込み先と書き込み先の各ファイルを丸かっこで囲んで指定します。

この例では、各 DATA ステップで同一の集約記憶域に格納されている外部ファイル(REGION1 と REGION2)を読み込みます。集約記憶域はファイル参照名 SALES を使用して参照されます。

```
filename sales 'aggregate-storage-location';
data total1;
  infile sales(region1);
  input machine $ jansales febsales marsales;
  totalsale=jansales+febsales+marsales;
run;
data total2;
  infile sales(region2);
  input machine $ jansales febsales marsales;
  totalsale=jansales+febsales+marsales;
run;
```

**例 4: PUT ステートメントの出力先の指定**

次の FILENAME ステートメントでは、ファイル参照名 OUT を動作環境固有のオプションを指定したプリンタに関連付けます。この FILENAME ステートメントでは、PUT ステートメントの出力をプリンタに送信するように指示します。

```
filename out printer operating-environment-option;
data sales;
  file out print;
  input salesrep $20. +6 jansales
    febsales marsales;
  put _infile_;
  datalines;
Jones, E. A.      124357 155321 167895
Lee, C. R.       111245 127564 143255
Desmond, R. T.   97631 101345 117865
;
```

FILENAME および FILE ステートメントを使用すると、同一セッションの実行中に PUT ステートメントの出力を複数のデバイスに送信できます。PUT ステートメントの出力内容をディスプレイモニタに送信するには、次に示すように FILENAME ステートメントで TERMINAL オプションを使用します。

```
filename show terminal;
data sales;
  file show;
```

```

input salesrep $20. +6 jansales
      febsales marsales;
put _infile_
datalines;
Jones, E. A.      124357 155321 167895
Lee, C. R.       111245 127564 143255
Desmond, R. T.   97631 101345 117865
;

```

### 例 5: 外部ファイル読み込み時のエンコードの指定

この例では、外部ファイルから SAS データセットを作成します。外部ファイルは Wlatin1 文字セットエンコーディングで、現在の SAS セッションは UTF-8 エンコーディングです。デフォルトでは、外部ファイルのエンコーディングがセッションエンコーディングと同じであるとみなします。そのため、文字データは新しいデータセットに正しく書き込まれません。

外部ファイルの読み込み時に使用するエンコーディングを指定するには、ENCODING=オプションを指定します。外部ファイルのエンコーディングを Wlatin1 に指定すると、新しい SAS データセットへの書き込み時に、外部ファイルが Wlatin1 から現在のセッションエンコーディングにトランスコードされます。これで、新しいデータセットにデータが UTF-8 のエンコーディングで正しく書き込まれるようになります。

```

libname myfiles 'SAS-library';

filename extfile 'external-file' encoding="wlatin1";
data myfiles.unicode;
  infile extfile;
  input Make $ Model $ Year;
run;

```

注: FILENAME ステートメントを使用して PROC CPORT で作成された移送ファイルを指定することはできません。移送ファイルをインポートするためには、移送元と移送先の SAS セッションのエンコーディングに互換性がなければなりません。

### 例 6: 外部ファイル書き込み時のエンコードの指定

この例では、SAS データセットから外部ファイルを作成します。現在のセッションエンコーディングは UTF-8 ですが、外部ファイルのエンコーディングは Wlatin1 にする必要があります。デフォルトでは、外部ファイルは現在のセッションエンコーディングを使用して書き込まれます。

外部ファイルへの書き込み時に使用するエンコーディングを指定するには、ENCODING=オプションを指定します。外部ファイルを Wlatin1 エンコーディングにするように指示すると、外部ファイルへの書き込み時に、UTF-8 から指定した Wlatin1 エンコーディングにデータがトランスコードされます。

```

libname myfiles 'SAS-library';
filename outfile 'external-file' encoding="wlatin1";

data _null_;
  set myfiles.cars;
  file outfile;
  put Make Model Year;
run;

```



**関連項目:**

- [“Using External Files and Devices” \(Batch and Line Mode Processing in SAS Viya\)](#)
- [“Printing and Routing Output” \(SAS Viya Universal Printing\)](#)

**ステートメント:**

- [“FILE ステートメント” \(75 ページ\)](#)
- [“%INCLUDE ステートメント” \(138 ページ\)](#)
- [“INFILE ステートメント” \(146 ページ\)](#)
- [“FILENAME ステートメント、SFTP アクセス方式” \(109 ページ\)](#)
- [“FILENAME ステートメント、URL アクセス方式” \(115 ページ\)](#)
- [“FILENAME ステートメント、ZIP アクセス方式” \(121 ページ\)](#)
- [“LIBNAME ステートメント” \(225 ページ\)](#)
- [“LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide](#)

**システムオプション:**

- [“LOCKDOWN System Option in SAS Intelligence Platform: Application Server Administration Guide](#)

---

**FILENAME ステートメント、SFTP アクセス方式**

SFTP プロトコルを使用してリモートファイルにアクセスできます。

**該当要素:** 任意の場所

**カテゴリ:** データアクセス

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

**構文**

**FILENAME** *fileref* SFTP '*external-file*' <*sftp-options*>;

**引数*****fileref***

有効なファイル参照名を指定します。

ヒ   ファイル参照名と外部ファイルの関連付けは、SAS セッション終了まで  
ン   維持されるか、または他の FILENAME ステートメントで関連付けの変  
ト   更や関連付けの取り消しを実行するまで維持されます。ファイルに対  
     するファイル参照名は必要に応じて何度でも変更できます。

---

**SFTP**

このアクセス方式を指定すると、SFTP(Secure File Transfer Protocol)を使用したファイルの読み込みや書き込みを、OpenSSH SSHD サーバーが稼働して

いるネットワーク上のホストコンピュータとの間で実行できるようになります。

### 'external-file'

読み込みまたは書き込みの対象となる外部ファイルの物理名を指定します。物理名には動作環境で判別できる名前を指定します。

**ヒント** ファイルの転送を実行するのではなく、ディレクトリリストの取得などのタスクを実行する場合は、外部ファイル名を指定する必要はありません。ファイル名のかわりに、引用符のみをステートメントに指定してください。

ファイル参照名には、1つのファイルまたは集約記憶域を関連付けることができます。

### sftp-options

ファイル属性や属性の処理など動作環境固有の詳細情報を指定します。

**動作環境** SFTP オプションの詳細については、各動作環境向けの SAS ドキュメントを参照してください。

参照項目: [“SFTP オプション” \(110 ページ\)](#)

### SFTP オプション

sftp-options には、次の値のいずれかを指定できます。

#### BATCHFILE='path'

SFTP コマンドを含むバッチファイルの絶対パスとファイル名を指定します。これらのコマンドは SFTP アクセス方式の実行時にサブミットされます。バッチファイルの処理が終了すると、SFTP 接続は終了します。

**要件** 指定するパスは一重引用符で囲む必要があります。

**ヒント** バッチファイルの処理が終了すると、SFTP 接続が終了するため、ファイル名の割り当てが利用できなくなります。後続の DATA ステップの処理で FILENAME SFTP ステートメントが必要になる場合は、FILENAME SFTP ステートメントを再度指定する必要があります。

例 [“例 4: バッチファイルを使用する” \(115 ページ\)](#)

#### CD='directory'

作業ディレクトリを変更するコマンドを発行します。この作業ディレクトリでは、指定した directory へのファイルの転送を実行します。

#### DEBUG

情報メッセージを SAS ログに書き込みます。

#### DIR

ディレクトリファイルにアクセスできるようにします。external-file 引数にディレクトリ名を指定してください。指定したホストに対して有効なディレクトリ構文を使用する必要があります。

**操作** CD オプションと DIR オプションは相互に排他的です。両方を指定すると、SFTP アクセス方式では CD オプションが無視されます。また、情報メッセージがログに書き込まれます。

ヒント SFTP を使用してディレクトリを作成する場合、DIR オプションと NEW オプションを組み合わせで使用します。ディレクトリがすでに存在する場合、NEW オプションは無視されます。

NEW オプションを使用せずに無効なディレクトリを指定すると、新しいディレクトリは作成されず、エラーメッセージが表示されます。

#### HOST='host'

この *host* には、OpenSSH SSHD サーバーが稼働しているリモートホストのネットワーク名を指定します。

ホスト名(例:**server.pc.mydomain.com**)またはコンピュータの IP アドレス(例:**2001:db8::**)のどちらかを指定できます。

#### LRECL=lrecl

この *lrecl* には、データの論理レコード長を指定します。

デフォルト 256

代わりに、“[LRECL= System Option](#)” ([SAS Viya System Options: Reference](#))を使用すると、グローバルな論理レコード長を指定できます。グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。

#### LS

SFTP サーバーに LS コマンドを発行します。LS オプションを指定すると、作業ディレクトリの内容がファイル属性を持たないレコードとして返されます。

制限事項 LS オプションでは、*.xAuthority* などのファイルの先頭にあるピリオドは表示されません。

操作 LS オプションと LSA オプションは相互に排他的です。両方のオプションを指定すると、LSA オプションが優先されます。

ヒント ファイルのサブセットのリストを返すには、LS オプションの他に LSFIL=オプションを使用します。

#### LSA

SFTP サーバーに LS コマンドを発行します。LSA オプションを指定すると、作業ディレクトリのすべての内容がファイル属性を持たないレコードとして返されます。

操作 LS オプションと LSA オプションは相互に排他的です。両方のオプションを指定すると、LSA オプションが優先されます。

*.xAuthority* などの先頭のピリオドを使用せずにファイルを表示するには、LS=オプションを使用します。

ヒント ファイルのサブセットのリストを返すには、LSA オプションの他に LSFIL=オプションを使用します。

#### LSFILE='character-string'

LS オプションと組み合わせで使用します。作業ディレクトリからファイルのサブセットのリストを要求するために使用する文字列を指定します。文字列は一重引用符で囲みます。

**制限事項** LSFILE=オプションは、LS または LSA オプションが指定されている場合にのみ使用できます。

**ヒント** 'character-string'の一部にワイルドカードを指定できます。

**例** 次のステートメントでは、*sales* で始まり、*sas* で終了するすべてのファイルをリストします。  

```
filename myfile sftp " ls lsfile='sales*.sas'
other-sftp-options;
```

### MGET

SFTP コマンドの MGET と同じように複数のファイルを転送します。

**ヒント** 転送処理全体が1つのファイルとして処理されます。ただし、新しいファイルの転送を開始するときに、EOV=変数が1に設定されます。

### NEW

DIR オプションの使用時に、SFTP でディレクトリを作成するように指定します。

**ヒント** ディレクトリがすでに存在する場合、NEW オプションは無視されます。

### OPTIONS='option-string'

ポート番号や詳細などの SFTP 構成オプションを指定します。

**注** OPTIONS 文字列の情報を隠す必要がある場合は、OPTIONSX オプションを使用します。

### OPTIONSX='option-string'

秘密鍵やパスフレーズなどの SFTP 構成オプションを指定します。option-string の情報はすべて、SAS ログへの書き込み時に隠されます。

**要件** OPTIONSX 文字列のパスフレーズにスペースが含まれる場合、そのパスフレーズを二重引用符で囲む必要があります。また、OPTIONSX 文字列は一重引用符で囲む必要があります。

**ヒント** パスフレーズは **-pw** パラメータを使用して渡されます。

### PATH

PATH または \$PATH などの検索パスにインストールされていない場合、SFTP 実行可能ファイルの場所を指定します。

**ヒント** PATH または \$PATH の検索パスからアクセス可能なディレクトリに、OpenSSH の "SFTP" 実行可能ファイルまたは PUTTY の "PSFTP" 実行可能ファイルをインストールすることをお勧めします。

### RECFM=recfm

この *recfm* には、次の2つのレコード形式のどちらかを指定します。

#### F

固定長レコード形式です。そのため、LRECL サイズのすべてのレコードには区切り文字となる改行が含まれていません。

**操作** グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用している場合、LRECL のデフォルト値は 256 になります。

## S

ストリームレコード形式です。データはイメージ(バイナリ)モードで転送されます。

**操作** 読み込むデータ量は、現在の LRECL の値または INFILE ステートメントに指定した NBYTE=変数の値で制御されます。NBYTE=オプションには、読み込まれるデータ量に等しくなる変数を指定します。このデータ量は、LRECL に指定した値に等しいか、それ以下の値にする必要があります。

**参照項目:** INFILE ステートメントの [NBYTE=オプション \(156 ページ\)](#)。

## V

可変長レコード形式(デフォルト設定)です。この形式ではレコードの長さが異なります。また、レコードは改行で区切られます。データはイメージ(バイナリ)モードで転送されます。

デフォルト V

### **USER='username'**

ユーザー名を指定します。

**ヒント** 公開鍵認証方式を使用する場合、Linux のホスト上では *username* を指定する必要はありません。

リモートの SSHD サーバーに接続する場合、SSH エージェントを使用する公開鍵認証方式を使用することをお勧めします。

### **WAIT\_MILLISECONDS=milliseconds**

SFTP の応答時間をミリ秒で指定します。

**デフォルト** 1,500 ミリ秒

**ヒント** タイムアウトメッセージがログに出力される場合は、WAIT\_MILLISECONDS オプションを使用して応答時間を長く設定します。

## 詳細

### 基本

SFTP(Secure File Transfer Protocol)は、ネットワーク上にある 2 つのホスト(クライアントとサーバー)間に対して保護された接続とファイル転送機能を提供します。2 つのホスト間のコマンドやデータは暗号化されます。クライアントマシンからリモートホスト(OpenSSH SSHD サーバー)との接続を開始します。

SFTP アクセス方式を使用すると、OpenSSH SSHD サーバーが稼働しているネットワーク上のホストコンピュータとの間でデータの読み込みや書き込みを実行できるようになります。クライアントアプリケーションとサーバーアプリケーションは同じコンピュータ上に存在していても、ネットワークで接続されている別のコンピュータ上に存在していてもかまいません。

実装の詳細については、OpenSSH SSHD サーバーのバージョンや設定方法によって異なります。

SFTP アクセス方式では、OpenSSH コマンドを使用してメッセージのデフォルトの送受信を行います。OpenSSH のカスタムインストールを実行してメッセージの設定を変更すると、SFTP アクセス方式を使用できない場合があります。

SFTP アクセス方式を使用するには、適切なクライアントソフトウェアをインストールする必要があります。SFTP アクセス方式は、the OpenSSH client on Linux では OpenSSH クライアントのみがサポートされます。

注: SFTP アクセス方式ではパスワード認証はサポートされません。

注: リモートの SSHD サーバーに接続する場合、SSH エージェントを使用する公開鍵認証方式を使用することをお勧めします。

注: SFTP アクセス方式の実行時に問題が発生する場合は、SAS システムを使用せずに SFTP クライアントから OpenSSH SSHD サーバーにアクセスできるかどうかを手動で検証してください。SAS システムを使用せずに SFTP クライアントの接続を手動で検証することにより、SSH または SSHD の構成や鍵認証が正しく設定されていることを確認できます。

### SFTP アクセス方式と SFTP プロンプト

SFTP アクセス方式では、次のプロンプトのみがサポートされます。プロンプトを変更すると、SFTP アクセス方式を使用できない場合があります。

- OpenSSH:

```
sftp>
sftp >
```

## 比較

SFTP の **get** コマンドや **put** コマンドと同じように、SFTP アクセス方式を使用するとファイルのアップロードやダウンロードを実行できます。ただし、この方式では最初にファイルをシステム上に保存せずに、SAS セッションに直接読み込みます。

## 例

### 例 1: SSHD サーバーの標準ポート接続

この例では、SSHD サーバーの標準ポートに接続した後、SFTP アクセス方式を使用して **test.dat** というファイルを読み込みます。

```
filename myfile sftp '/users/xxxx/test.dat' host="linuxhost1";
data _null_;
  infile myfile trunccover;
  input a $25.;
run;
```

### 例 2: SSHD サーバーの非標準ポート接続

この例では、SSHD サーバーのポート番号 4117 に接続した後、SFTP アクセス方式を使用して **test.dat** というファイルを読み込みます。

```
filename myfile sftp '/users/xxxx/test.dat' host="linuxhost1" options="-oPort=4117";
data _null_;
  infile myfile trunccover;
  input a $25.;;
```

```
run;
```

### 例 3: リモートホストのディレクトリからファイルを読み込む

次の例では、リモートホスト上のディレクトリからファイル **test.dat** と **test2.dat** を読み込みます。

```
filename infile sftp '/users/xxxx/' host="linuxhost1" dir;
data _null_;
  infile infile(test.dat) truncover;
  input a $25.;
  infile infile(test2.dat) truncover;
  input b $25.;
run;
```

### 例 4: バッチファイルを使用する

次の例では、INFILE ステートメントを処理するとき、FILENAME SFTP ステートメントに関連付けられたバッチファイル **sftpcmds** が実行されます。

```
filename process sftp '' host="linuxhost1" user="userid"
  batchfile="c:/stfpdir/sftpcmds.bat";
data _null_;
  infile process;
run;
```

### 関連項目:

- Barrett, Daniel J., Richard E. Silverman, and Robert G. Byrnes. 2005. "SSH, The Secure Shell: The Definitive Guide." Sebastopol, CA: O'Reilly

### ステートメント:

- ["FILENAME ステートメント" \(96 ページ\)](#)
- ["FILENAME ステートメント、URL アクセス方式" \(115 ページ\)](#)
- ["FILENAME ステートメント、ZIP アクセス方式" \(121 ページ\)](#)
- ["LIBNAME ステートメント" \(225 ページ\)](#)

---

## FILENAME ステートメント、URL アクセス方式

URL アクセス方式を使用してリモートファイルにアクセスできます。

**該当要素:** 任意の場所

**カテゴリ:** データアクセス

**制限事項:** このステートメントは CAS Server ではサポートされていません。

SAS がロックダウン状態にある場合、FILENAME ステートメントの URL アクセス方式は使用できません。サーバー管理者は、このアクセス方式がロックダウン状態でも使用できるように、同方式を再有効化できます。FILENAME ステートメントの URL アクセス方式が再有効化されると、SOAP プロシジャは自動的に再有効化されます。

---

### 構文

```
FILENAME fileref URL 'external-file'<url-options>;
```

## 引数

### ***fileref***

有効なファイル参照名を指定します。

ヒ ファイル参照名と外部ファイルの関連付けは、SAS セッション終了まで  
ン 維持されるか、または他の FILENAME ステートメントで関連付けの変  
ト 更や関連付けの取り消しを実行するまで維持されます。ファイルに対  
するファイル参照名は必要に応じて何度でも変更できます。

### **URL**

このアクセス方式を指定すると、URL サーバーが稼働しているネットワーク上のホストコンピュータとの間でファイルの読み込みや書き込みを実行できるようになります。

別名 HTTP

### **'external-file'**

URL サーバーからの読み込みや URL サーバーへの書き込みの対象となるファイルの名前を指定します。ファイルへのアクセスには、TLS (Transport Layer Security) プロトコル `https` も使用できます。ファイルは次のいずれかの形式で指定する必要があります。

- `http://hostname/file`
- `https://hostname/file`
- `http://hostname:portno/file`
- `https://hostname:portno/file`

### **URL オプション**

`url-options` には、次の値のいずれかを指定できます。

#### **ACCEPT='header-type'**

Accept:ヘッダーを指定します。Accept:ヘッダーを使用すると、レスポンスとして受け入れ可能である特定のメディアタイプを指定できます。

デフォルト \*/\*

要件 `header-type` は一重引用符または二重引用符で囲む必要があります。

#### **BLOCKSIZE=blocksize**

この `blocksize` には、URL データバッファのサイズをバイト単位で指定します。

デフォルト 8K

#### **CONNECT**

プロキシを通して URL にアクセスしている時に、クライアントとプロキシ間の接続とプロキシとサーバー間の接続を作成します。

要件 PROXY=オプションは CONNECT オプションと一緒に使用してください。PROXY=オプションなしで CONNECT オプションを使用した場合、接続されません。

操作 "http"を `external-file` 引数内で使用すると、接続はされますが TLS プロトコルは使用されません。



参照項目: [“PROXY=url” \(118 ページ\)](#)

## DEBUG

デバッグ情報を SAS ログに書き込みます。

ヒント HELP コマンドの実行結果がレコードとして返されます。

## HEADERS=*fileref*

URL アクセス方式を使用してファイルを開くときにヘッダー情報の書き込み先となるファイル参照名を指定します。このヘッダー情報は、SAS ログに書き込まれる情報と同じです。

要件 ファイル参照名は、先行する FILENAME ステートメントで定義する必要があります。

操作 DEBUG オプションを指定せずに HEADERS=オプションを指定しても、DEBUG オプションは自動的に有効になります。

デフォルトでは、ログ情報は上書きされます。ログ情報を追加するには、ファイル参照名を作成する FILENAME ステートメントに MOD オプションを指定する必要があります。

## LRECL=*lrecl*

この *lrecl* には、データの論理レコード長を指定します。

デフォルト 256

操作 かわりに、“[LRECL= System Option](#)” ([SAS Viya System Options: Reference](#))を使用すると、グローバルな論理レコード長を指定できません。グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用している場合、LRECL のデフォルト値は 256 になります。

## PASS=*password*

この *password* には、USER オプションで指定したユーザー名とともに使用するパスワードを指定します。

ヒント PASS オプションのかわりに PROMPT オプションを指定できます。PROMPT オプションは、システムに対してパスワードの入力を求めるプロンプトを表示するように指示します。

暗号化されたパスワードを使用する場合、テキスト文字列を隠すために PWENCODE プロシジャを使用します。次に、暗号化されたパスワードを PASS=オプションに入力します。詳細については、“[PWENCODE](#)” ([SAS Viya Data Management and Utility Procedures Guide](#))を参照してください。

## PPASS=*password*

この *password* には、PUSER オプションで指定したユーザー名とともに使用するパスワードを指定します。この PPASS オプションは、プロキシサーバーへアクセスするために使用します。

ヒント PPASS オプションのかわりに PROMPT オプションを指定できます。PROMPT オプションは、システムに対してパスワードの入力を求めるプロンプトを表示するように指示します。

暗号化されたパスワードを使用する場合、テキスト文字列を隠すために PWENCODE プロシジャを使用します。次に、暗号化されたパスワードを PASS=オプションに入力します。詳細については、“PWENCODE” ([SAS Viya Data Management and Utility Procedures Guide](#))を参照してください。

**PROMPT**

必要に応じて、ユーザーのログインパスワードの入力を求めるプロンプトを表示するように指定します。

ヒント PROMPT=を指定する場合、PASS=または PPASS=を指定する必要はありません。

**PROXY=url**

プロキシサーバーの URL(Uniform Resource Locator)を次のどちらかの形式で指定します。

`http://hostname/`

`http://hostname:portno/`

参照項目: [“CONNECT” \(116 ページ\)](#)

**PUSER='username'**

この *username* は、URL プロキシサーバーにログオンするために使用されません。

操作 PUSER オプションを指定すると、プロキシサーバーの設定の有無にかかわらず、USER オプションは Web サーバーに渡されます。

PROMPT を指定して PUSER を指定しない場合、ID とパスワードの入力を求めるプロンプトがユーザーに表示されます。

ヒント PUSER='\*'と指定すると、ID の入力を求めるプロンプトがユーザーに表示されます。

**RECFM=recfm**

この *recfm* には、次の 3 つのレコード形式のいずれかを指定します。

**F**

固定長レコード形式です。そのため、LRECL サイズのすべてのレコードには区切り文字となる改行が含まれていません。データはイメージ(バイナリ)モードで転送されます。

操作 グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。

**S**

ストリームレコード形式です。データはイメージ(バイナリ)モードで転送されます。

別名 N

ヒント 読み込むデータ量は、現在の LRECL の値または INFILE ステートメントに指定した NBYTE=変数の値で制御されます。NBYTE=オプションには、読み込まれるデータ量に等しくなる変数を指定し

ます。このデータ量は、LRECL に指定した値に等しいか、それ以下の値にする必要があります。

参照項目: INFILE ステートメントの [NBYTE=オプション \(156 ページ\)](#)。

## V

可変長レコード形式(デフォルト設定)です。この形式では、レコードの長さが異なります。また、レコードはテキスト(ストリーム)モードで転送されます。

ヒント LRECL の値よりも大きいレコードは切り捨てられます。

デフォルト V

### TERMSTR='eol-char'

ここで、*eol-char* には、RECFM=V を指定した場合に使用する改行文字を指定します。次の 4 つの値を使用できます。

CR      キャリッジリターン(CR)  
 CRLF    ラインフィード(LF)が後に続くキャリッジリターン(CR)  
 LF      ラインフィードのみ(デフォルト設定)  
 NULL    NULL 文字(0x00)

デフォルト LF

制限事項 このオプションは RECFM=V を指定した場合にのみ使用します。

### USER='username'

この *username* は URL サーバーにログオンするために使用されます。

操作 USER オプションを指定して PUSER オプションを指定しない場合、USER オプションの処理はプロキシサーバーの指定によって異なります。プロキシサーバーを指定していない場合、USER オプションは Web サーバーに渡されます。プロキシサーバーを指定している場合、USER オプションは Web サーバーに渡されます。

PUSER オプションを指定すると、プロキシサーバーの設定の有無にかかわらず、USER オプションは Web サーバーに渡されます。

PROMPT を指定して USER または PUSER を指定しない場合、ID とパスワードの入力を求めるプロンプトがユーザーに表示されます。

ヒント USER='\*'と指定すると、ID の入力を求めるプロンプトがユーザーに表示されます。

## 詳細

URL が“http”ではなく“https”で始まる場合は、TLS (Transport Layer Security) プロトコルが使用されます。TLS とその先行プロトコルである SSL (Secure Sockets Layer) は、インターネット経由で通信セキュリティを提供するために設計された暗号化プロトコルです。また、TLS および SSL プロトコルは、ネットワークデータのプライバシー、データ整合性、認証を提供します。さらに、TLS は、暗号化サービスを提供する以外にも、クライアントおよびサーバー認証を実

行するほか、メッセージ認証コードを使用してデータ整合性を確保します。TLS プロトコルを使うと、クライアント/サーバーアプリケーションは、盗聴や改ざんを防ぐために設計された方法を通じて、ネットワーク通信が行えるようになります。TLS は、すべての主要ブラウザでサポートされています。

注: TLS に関する説明はすべて、先行プロトコルである SSL に対しても適用されます。

#### 動作環境の情報

FILENAME ステートメントを使用する場合は、動作環境固有の情報が必要になります。URL アクセス方式はここで完全に説明されていますが、ファイル名の指定方法に関する詳細は、お使いの動作環境向けの SAS ドキュメントを参照してください。

## 例

### 例 1: Web サイトのファイルへのアクセス

この例では、サイト **www.a.com** のドキュメント **test.dat** にアクセスします。

```
filename foo url 'http://www.a.com/test.dat'
  proxy='http://www.gt.sas.com';
```

### 例 2: ユーザー ID とパスワードの指定

次の例では、TLS プロトコルを使用してサイト **www.b.com** 上のドキュメント **file1.html** にアクセスします。ユーザー ID とパスワードを入力する必要があります。

```
filename foo url 'https://www.b.com/file1.html'
  user='jones' prompt;
```

### 例 3: URL ファイルから先頭 15 レコードを読み込む

この例では、URL ファイルから先頭の 15 レコードを読み込んだ後、そのレコードを PUT ステートメントを使用して SAS ログに書き込みます。

```
filename foo url
  'http://support.sas.com/techsup/service_intro.html';

data _null_;
  infile foo length=len;
  input record $varying200. len;
  put record $varying200. len;
  if _n_=15 then stop;
run;
```

## 関連項目:

- [“Transport Layer Security \(TLS\)” \(Encryption in SAS Viya\)](#)

## ステートメント:

- [“FILENAME ステートメント” \(96 ページ\)](#)
- [“FILENAME ステートメント、SFTP アクセス方式” \(109 ページ\)](#)
- [“FILENAME ステートメント、ZIP アクセス方式” \(121 ページ\)](#)
- [“LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide](#)

## FILENAME ステートメント、ZIP アクセス方式

ZIP ファイルにアクセスできるようにします。

**該当要素:** 任意の場所

**カテゴリ:** データアクセス

**制限事項:** このステートメントは CAS Server ではサポートされていません。

**注:** ZIP アクセス方式では、WinZip ファイル圧縮で作成されたファイルのみ読み込みと書き込みが行われます。

### 構文

```
FILENAME fileref ZIP 'external-file'<zip-options>;
```

### 引数

#### *fileref*

有効なファイル参照名を指定します。

ヒ    ファイル参照名と外部ファイルの関連付けは、SAS セッション終了まで  
ン    維持されるか、または他の FILENAME ステートメントでファイル参照  
ト    名の変更や取り消しを実行するまで維持されます。ファイルに対する  
      ファイル参照名は必要に応じて何度でも変更できます。

#### ZIP

このアクセス方式を指定すると、ZIP ファイルが使用できるようになります。

#### '*external-file*'

読み込みまたは書き込みの対象となる ZIP ファイル名を指定します。

動    外部ファイルの物理名を指定する方法の詳細については、各動作環境向  
作    けの SAS ドキュメントを参照してください。  
環  
境

注    DATA ステップで同じ ZIP ファイルのエントリを複数回指定すると、エ  
      ラーになります。複数のエントリは重なり合ってしまう、予想外の結果  
      が生じます。

ヒ    外部ファイルにファイル参照名を割り当てる場合は、*external-file* を指  
ン    定します。ファイル参照名には、MEMBER=構文を使用して 1 つのファ  
ト    イル、または *fileref(member)* 構文を使用して集約記憶域を関連付けるこ  
      とができます。

MEMBER=構文ではワイルドカードを使用できません。アスタリスク (\*) はゼロ以上の文字と一致します。疑問符 (?) は 1 文字と一致します。ワイルドカードは、エントリの読み込み時に存在アクションについてサポートされます。ワイルドカードは、書き込みまたは削除アクションについてはサポートされません。たとえば、エントリ "A\*" を書き込むと、エントリ "A\*" が作成されます。"A\*" というエントリを削除すると、"A\*" は削除されますが、"A" で始まるエントリはいずれも削除されません。"A\*\*

を指定して exist 関数を呼び出すと、“A”で始まるエントリが1つ以上存在すれば真が返されます。

## ZIP オプション

zip-options には、次の値のいずれかを指定できます。

### COMMENT="comment-string"

ZIP ファイルに情報のコメントを書き込みます。

### COMPRESSION='compression-level'

ZIP ファイルメンバへの書き込み時に使用する圧縮レベルを指定します。圧縮レベルの有効な値は 0 から 9 です。0 の値では、圧縮なしでファイルを保存します。値 9 は最大の圧縮レベルを意味します。

デフォルト 6

制限事項 COMPRESSON=は書き込むためにファイルを開く際にだけ使用されます。

## DEBUG

デバッグ情報を SAS ログに書き込みます。

### ENCODING=encoding-value

外部ファイルからの読み込みや外部ファイルへの書き込みに使用するエンコーディングを指定します。ENCODING=の値は、外部ファイルのエンコーディングが現在のセッションエンコーディングとは異なることを示しています。

外部ファイルからデータを読み込む場合は、指定したエンコーディングからセッションエンコーディングにデータがトランスコードされます。外部ファイルにデータを書き込む場合は、セッションエンコーディングから指定したエンコーディングにデータがトランスコードされます。

デフォルト SAS では、外部ファイルのエンコーディングがセッションエンコーディングと同じであるとみなします。

参照項目: [“Encoding Values in SAS Language Elements” \(SAS Viya National Language Support: Reference Guide\)](#)

## FILEEXT

ファイル拡張子が存在しない場合にファイル名の番号に拡張子が自動的に追加されるように指定します。

操作 自動呼び出しマクロ機能では、拡張子.SAS が常にファイルアクセス方式に渡されます。この拡張子は自動呼び出しマクロライブラリ内のファイルを開くときに使用されます。DATA ステップでは、拡張子.DATA が常に渡されます。自動呼び出しマクロライブラリに対してファイル参照名を定義し、そのライブラリにあるファイルの拡張子が.SAS の場合、FILEEXT オプションを使用します。ライブラリに拡張子を持つファイルが存在しない場合は、FILEEXT オプションを使用しないでください。たとえば、DATA ステップで入力ファイルにファイル参照名を定義し、そのファイル X の拡張子が.DATA の場合、ファイル X.DATA を読み込むために FILEEXT オプションを使用します。INFILE または FILE ステートメントを使用する場合は、大文字と小文字を維持するためにメンバ名と拡張子を引用符で囲みます。

ヒント INFILE または FILE ステートメントでファイルの拡張子を指定すると、FILEEXT オプションは無視されます。

参照項目: ["LOWCASE\\_MEMNAME" \(123 ページ\)](#)

### LOWCASE\_MEMNAME

自動呼び出しマクロを使用して、ZIP ファイルから小文字のディレクトリ名やメンバ名を取得できるようにします。

制限事項 SAS 自動呼び出しマクロを使用して名前を取得するときは、常に大文字のディレクトリメンバ名を探します。大文字と小文字が混在するディレクトリ名やメンバ名はサポートされていません。

参照項目: ["FILEEXT" \(122 ページ\)](#)

### LRECL=*lrecl*

この *lrecl* には、データの論理レコード長を指定します。

デフォルト 32767

操作 かわりに、["LRECL= System Option" \(SAS Viya System Options: Reference\)](#)を使用すると、グローバルな論理レコード長を指定できます。グローバル LRECL システムオプションのデフォルト値は 32767 です。

### NAMEENCODING=*encoding-value*

ZIP ファイルのエントリ名とコメントに使用するエンコーディングを指定します。NAMEENCODING=の値は、エントリ名とコメントのエンコーディングが、現在のセッションのエンコーディングとは異なることを示します。

デフォルト コードページ 437

例 filename zs zip "xyz.zip" nameencoding=sjis member="s" termstr=lf;

### RECFM=*recfm*

この *recfm* には、次の 4 つのレコード形式のいずれかを指定します。

**F** 固定長レコード形式です。各レコードは同じ長さになります。

**N** バイナリ形式です。レコード境界なしでストリームバイトからなるファイルです。

**S** ストリームレコード形式です。

操作 読み込むデータ量は、現在の LRECL の値または INFILE ステートメントに指定した NBYTE=変数の値で制御されます。NBYTE=オプションには、読み込まれるデータ量に等しくなる変数を指定します。このデータ量は、LRECL に指定した値に等しいか、それ以下の値にする必要があります。



参照 INFILE ステートメントの [NBYTE=オプション \(156 ページ\)](#)。  
項目:

## V

可変長レコード形式(デフォルト設定)です。この形式では、レコードの長さが異なります。また、レコードはテキスト(ストリーム)モードで転送されます。

操作 LRECL の値よりも大きいレコードは切り捨てられます。

デフォルト V

### TERMSTR='eol-termination-character'

終端文字は、読み込み操作では改行文字で、レコードの書き込み操作では終端文字になります。次の 4 つの値を使用できます。

CR      キャリッジリターン(CR)  
LF      ラインフィードのみ(デフォルト設定)  
NULL    NULL 文字(0x00)

デフォルト LF

## 例

### 例 1: ディレクトリから ZIP ファイルメンバの読み込み

次の例では、ZIP ファイルメンバ *test1.txt* を ZIP ファイル **testzip** から読み込みます。

```
filename foo ZIP 'U:\directory1\testzip.zip' member="test1.txt";
```

```
data _null_;
infile foo;
input a $80.;
run;
```

### 例 2: ZIP ファイルの新ディレクトリメンバへの書き込み

次の例ではファイル *shoes* を ZIP ファイル **testzip** に書き込みます。

```
filename foo ZIP 'U:\directory1\testzip.zip';
```

```
data _null_;
file foo(shoes);
set sashelp.shoes;
put region $25. product $14.;
run;
```

### 例 3: ディレクトリメンバからの読み込み

この例では、ファイル *shoes* を ZIP ファイル **testing1** から読み込みます。

```
filename foo ZIP 'U:\directory1\testzip.zip';
```

```
data shoes;
length region $25 product $14;
```



```
infile foo(shoes);
input region $25. product $14.;
run;
```

## 関連項目:

### ステートメント:

- [“FILENAME ステートメント” \(96 ページ\)](#)
- [“FILENAME ステートメント、SFTP アクセス方式” \(109 ページ\)](#)
- [“FILENAME ステートメント、URL アクセス方式” \(115 ページ\)](#)

---

## FOOTNOTE ステートメント

プロシジャまたは DATA ステップの出力の下部にテキストを 10 行まで書き込みます。

**該当要素:** 任意の場所

**カテゴリ:** 出力制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。  
FOOTNOTE ステートメントは、Unicode をサポートしていません。

**要件** FILE ステートメントを使用する場合は、FOOTNOTE オプションを指定する必要があります。

## 構文

```
FOOTNOTE <n> <'text' | "text">;
```

### 引数なし

引数を指定せずに FOOTNOTE ステートメントを使用すると、現在設定されているフットノートをすべて取り消します。

### 引数

*n*

フットノートを表示する相対行番号を指定します。

**デフォルト** *n* を省略した場合、SAS は 1 の値とみなします。

**範囲** *n* には 1 から 10 まで指定できます。

**ヒント** フットノートの場合、行は 1 から順に表示されます。番号が最大の FOOTNOTE ステートメントは、一番下の行に表示されます。

### *ods-format-options*

ODS HTML、RTF、および PRINTER(PDF)の各出力先に対して形式オプションを指定します。

### **BOLD**

フットノートのテキストを太字で表示するように指定します。

ODS 出力先 HTML、RTF、PRINTER

---

**COLOR=***color*

フットノートのテキストの色を指定します。

別名 C

---

ODS 出力先 HTML、RTF、PRINTER

---

例 “例 3: ODS (Output Delivery System)を使用したタイトルとフットノートのカスタマイズ” (405 ページ)

---

**BCOLOR=***color*

フットノート表示部分の背景色を指定します。

ODS 出力先 HTML、RTF、PRINTER

---

**FONT=***font-face*

使用するフォントを指定します。複数のフォントを指定した場合、出力先デバイスはシステムにインストールされている最初のフォントを使用します。

別名 F

---

ODS 出力先 HTML、RTF、PRINTER

---

**HEIGHT=***size*

ポイントサイズを指定します。

別名 H

---

ODS 出力先 HTML、RTF、PRINTER

---

例 “例 3: ODS (Output Delivery System)を使用したタイトルとフットノートのカスタマイズ” (405 ページ)

---

**ITALIC**

フットノートのテキストをイタリック体で表示するように指定します。

ODS 出力先 HTML、RTF、PRINTER

---

**JUSTIFY=** CENTER | LEFT | RIGHT

配置する位置を指定します。

**CENTER**

中央揃えで表示するように指定します。

別名 C

---

**LEFT**

左寄せで表示するように指定します。

別名 L

---

**RIGHT**

右寄せで表示するように指定します。

別名 R

---

別名	J
ODS 出力先	HTML、RTF、PRINTER
例	<a href="#">“例 3: ODS (Output Delivery System)を使用したタイトルとフットノートのカスタマイズ” (405 ページ)</a>

**LINK='url'**

ハイパーリンクを指定します。

ODS 出力先	HTML、RTF、PRINTER
ヒント	LINK=の表示プロパティは、常に現在のスタイルから取得します。

**UNDERLIN= 0 | 1 | 2 | 3**

後ろに指定するテキストに下線を表示するかどうかを指定します。0 を指定すると下線は表示されません。1、2、3 を指定すると下線が表示されます。

別名	U
ODS 出力先	HTML、RTF、PRINTER
ヒント	ODS では値 1、2、3 に対して同じ種類の下線が生成されます。SAS/GRAPH では、値 1、2、3 に従って生成される下線の幅が徐々に太くなります。

**注** デフォルトでは、ODS での FOOTNOTE ステートメントの表示設定は、現在のスタイルのシステムフットノートに関連するスタイル要素から取得します。FOOTNOTE ステートメントの構文で *ods-format-options* を指定すると、現在のスタイルから提供される設定より優先されます。現在のスタイルは、ODS 出力先によって異なります。

**ヒ** これらのオプションを文字、1 単語、複数単語ごとに指定することができます。オプションに続けて、*text* にそれぞれの文字や単語を指定します。たとえば、次のコードでは、タイトルテキスト“Red, White, and Blue”が異なる色で表示されます。

```
footnote color=red "Red," color=white "White, and" color=blue "Blue";
```

**'text' | "text"**

フットノートとして表示するテキストを一重引用符または二重引用符で囲んで指定します。

**ヒ** 以前のリリースとの互換性を保つため、引用符で囲まれていない一部のテキストが受け入れられます。新しいプログラムを作成したり、既存のプログラムを更新する場合は、指定するテキストを *常に* 引用符で囲むようにしてください。

マクロやマクロ変数を使用すると、FOOTNOTE ステートメント内の情報を変更できます。フットノートが二重引用符(“)で囲まれている場合、指定されたテキストはそのフットノートに置き換えられます。フットノートが一重引用符(“)で囲まれている場合、テキストは置き換えられません。

間に空白を含まない一重引用符(')または二重引用符(“)の記号のみをテキスト文字列として指定すると、一重引用符(')または二重引用符(“)がそれぞれ生成されます。

## 詳細

FOOTNOTE ステートメントはステップまたはステップに関連付けられた RUN グループの実行時に有効になります。フットノートを指定すると、指定したフットノートをキャンセルするか再度設定するまで、すべてのページに同じフットノートが表示されます。行番号を指定して FOOTNOTE ステートメントを実行すると、指定した行にあらかじめ設定されていた FOOTNOTE ステートメントは取り消されます。また、指定した行以降のすべてのフットノートが取り消されません。

### 動作環境の情報

対話モードの場合、タイトルの最大長は 254 文字です。指定されたフットノートの長さが LINESIZE オプションの値よりも大きい場合、そのフットノートは行のサイズに収まるように切り捨てられます。

## 比較

フットノートは、**FOOTNOTES** ウィンドウでも作成できます。詳細については、該当するウィンドウのオンラインヘルプを参照してください。

ODS(Output Delivery System)を使用してフットノートを変更することができます。“[例 3: ODS \(Output Delivery System\)を使用したタイトルとフットノートのカスタマイズ](#)” (405 ページ)を参照してください。

## 例: FOOTNOTE ステートメントを使用する

これらの FOOTNOTE ステートメントの例では、どちらも同じフットノートが表示されます。

- footnote8 "Managers' Meeting";
- footnote8 'Managers' Meeting';

これらの FOOTNOTE ステートメントの例では、ODS HTML、RTF、PRINTER(PDF)の各出力先に対して、形式オプションを使用します。詳細な例については、“[例 3: ODS \(Output Delivery System\)を使用したタイトルとフットノートのカスタマイズ](#)” (405 ページ)を参照してください。

```
footnote j=left height=20pt
  color=red "Prepared "
  c='#FF9900' "on";
footnote2 j=center color=blue
  height=24pt "&SYSDATE9";
footnote3 link='http://support.sas.com' "SAS";
```

## 関連項目:

### ステートメント:

- “[TITLE ステートメント](#)” (399 ページ)

## FORMAT ステートメント

変数に出力形式を関連付けます。

**該当要素:** DATA ステップまたは PROC ステップ

**カテゴリ:** CAS  
情報

**種類:** 宣言

### 構文

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

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

**FORMAT** *variable-1* <...*variable-n*> *format variable-1* <...*variable-n*> *format*;

### 引数

#### *variable*

1 つまたは複数の変数に出力形式を関連付けます。少なくとも 1 つの *variable* を指定する必要があります。

ヒ 変数から出力形式の関連付けを取り消すには、DATA ステップまたは  
ン PROC DATASETS 内で、この変数を FORMAT ステートメントに出力形  
ト 式を指定せずに使用します。DATA ステップでは、この FORMAT ステ  
ートメントを SET ステートメントの後に配置します。“[例 3: 出力形式の  
取り消し](#)” (132 ページ) を参照してください。PROC DATASETS も使用  
することができます。

#### *format*

変数の値を出力するときに適用する出力形式を指定します。

ヒント FORMAT ステートメントを使用して変数に関連付けられた出力形式  
は、コロン修飾子で使われる出力形式と同じように、後続の PUT  
ステートメントで動作します。コロン修飾子の使用方法の詳細につ  
いては、“[PUT ステートメント、リスト](#)” (343 ページ) を参照してく  
ださい。

参照項 [SAS Viya Formats and Informats: Reference](#)  
目:

#### DEFAULT=*default-format*

FORMAT ステートメントにリストされていない変数の値を表示するときに  
使用する一時的なデフォルトの出力形式を指定します。このデフォルトの出  
力形式は、現在の DATA ステップにのみ適用されます。出力内の変数に常  
に関連付けられるわけではありません。

DEFAULT= に指定した出力形式の適用対象

- FORMAT ステートメントまたは ATTRIB ステートメントに指定されてい  
ない変数
- SAS データセット内に常に関連付けられてる出力形式がない変数

- 出力形式を明示的に指定して出力されていない変数

**デフォルト** DEFAULT=の指定を省略すると、デフォルトの数値の出力形式として BESTw. を使用し、デフォルトの文字の出力形式として \$w. をそれぞれ使用します。

**制限事項** このオプションは DATA ステップ内でのみ使用できます。

**ヒント** DEFAULT=は、FORMAT ステートメントのどの位置に指定してもかまいません。数値に対するデフォルト値、文字に対するデフォルト値、またはその両方を指定できます。

**例** “例 1: 出力形式とデフォルト値を割り当てる” (130 ページ)

## 詳細

FORMAT ステートメントでは、標準の SAS 出力形式または PROC FORMAT に定義したユーザー定義の出力形式を使用できます。1 つの FORMAT ステートメントでは、同じ出力形式を複数の変数に関連付けることができます。また、変数ごとに異なる出力形式を関連付けることもできます。同じ変数が複数の FORMAT ステートメント内に指定されている場合、最後に関連付けられた出力形式が使用されます。

変数に出力形式を常に割り当てるようにするには、DATA ステップ内で FORMAT ステートメントを使用します。ステートメントを使用すると、指定した変数が含まれる SAS データセットのディスクリプタ情報が変更されます。一部の PROC ステップでも FORMAT ステートメントを使用できますが、ルールは異なります。詳細については、[SAS Viya Data Management and Utility Procedures Guide](#) を参照してください。

## 比較

ATTRIB ステートメントと FORMAT ステートメントは、どちらも変数に出力形式を関連付けることができます。また、どちらのステートメントを使用しても変数に関連付けられた出力形式を変更することができます。変数に関連付けられた出力形式の変更や取り消しを行うには、PROC DATASETS 内で FORMAT ステートメントを使用します。

## 例

### 例 1: 出力形式とデフォルト値を割り当てる

この例では、FORMAT ステートメントを使用して数値変数と文字変数に出力形式とデフォルトの出力形式を割り当てます。デフォルトの出力形式はデータセット内の変数に関連付けられません。しかし、現在の DATA ステップで PUT ステートメントを使用して変数を出力するときに、指定した内容が反映されます。

```
data tstfmt;
  format W $char3.
  Y 10.3
  default=8.2 $char8.;
  W='Good morning.';
  X=12.1;
  Y=13.2;
  Z='Howdy-doody';
  put W/X/Y/Z;
```

```
run;
proc contents data=tstfmt;
run;
proc print data=tstfmt;
run;
```

次の出力結果は、PROC CONTENTS の実行結果の一部と PROC PRINT で生成したレポートを示しています。

**アウトプット 2.3** PROC CONTENTS の実行結果の一部と PROC PRINT で生成したレポート

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
1	W	Char	3	\$CHAR3.
3	X	Num	8	
2	Y	Num	8	10.3
4	Z	Char	11	

**アウトプット 2.4** PROC PRINT で生成したレポート

Obs	W	Y	X	Z
1	Goo	13.200	12.1	Howdy-doodo

デフォルトの出力形式が変数 X と Z に適用され、指定した出力形式が変数 W と Y に適用されています。

PUT ステートメントの実行結果は次のようになります。

```
----+----1----+----2
Goo
12.10
13.200
Howdy-do
```

**例 2: 1 つの出力形式を複数の変数に関連付ける**

この例では、FORMAT ステートメントを使用して 1 つの出力形式を複数の変数に関連付けます。

```
data report;
  input Item $ 1-6 Material $ 8-14 Investment 16-22 Profit 24-31;
  format Item Material $upcase9. Investment Profit dollar15.2;
  datalines;
shirts cotton 2256354 83952175
ties silk 498678 2349615
suits silk 9482146 69839563
belts leather 7693 14893
shoes leather 7936712 22964
;
```

```
run;
options pageno=1 nodate ls=80 ps=64;
proc print data=report;
  title 'Profit Summary: Kellam Manufacturing Company';
run;
```

**アウトプット 2.5** 1つの出力形式を複数の変数に関連付けた後の出力結果

Profit Summary: Kellam Manufacturing Company				
Obs	Item	Material	Investment	Profit
1	SHIRTS	COTTON	\$2,256,354.00	\$83,952,175.00
2	TIES	SILK	\$498,678.00	\$2,349,615.00
3	SUITS	SILK	\$9,482,146.00	\$69,839,563.00
4	BELTS	LEATHER	\$7,693.00	\$14,893.00
5	SHOES	LEATHER	\$7,936,712.00	\$22,964.00

### 例 3: 出力形式の取り消し

この例では、SAS データセットにある変数から指定した出力形式の関連付けを取り消します。FORMAT ステートメントと SET ステートメントを指定する順序が重要です。

```
data rtest;
  set rtest;
  format x;
run;
```

### 関連項目:

- SAS Viya Utilities Procedures Guide 内の DATSETS プロシジャ

### ステートメント:

- ["ATTRIB ステートメント" \(29 ページ\)](#)

---

## GO TO ステートメント

指定したステートメントラベルにプログラム実行を移動します。RETURN ステートメントが後に続く場合、DATA ステップの先頭に実行を戻します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 実行

**別名:** GOTO

---



## 構文

GO TO *label*;

## 引数

### *label*

GO TO の移動先となるステートメントラベルを指定します。移動先は同じ DATA ステップ内になければなりません。*label* 引数は必ず指定する必要があります。

## 比較

GO TO ステートメントと LINK ステートメントは似ています。GO TO ステートメントは RETURN ステートメントを指定せずに使用できますが、通常、LINK ステートメントでは RETURN ステートメントを明示的に指定して使用します。GO TO ステートメントと LINK ステートメントでは、後に続く RETURN ステートメントのアクションが異なります。LINK ステートメントの後に RETURN ステートメントを指定する場合は、LINK ステートメントの直後のステートメントに実行を戻します。GO TO ステートメントの後に RETURN ステートメントを指定し、GO TO ステートメントの前に LINK ステートメントがない場合は、DATA ステップの先頭に実行を戻します。このとき、GO TO ステートメントの前に LINK ステートメントがある場合は、LINK ステートメントの直後の最初のステートメントから実行を続けます。

GO TO ステートメントは、DO-END および IF-THEN/ELSE のプログラムロジックに置き換えることができます。

## 例: GO TO ステートメントで RETURN ステートメントを使用する

次に示す GO TO ステートメントを使用します。

- この例では、条件が真になる場合、GO TO ステートメントは ADD という名前のラベルに移動し、そこから実行を続けます。条件が偽になる場合、PUT ステートメントを実行し、GO TO ラベルに指定されているステートメントを実行します。

```
data info;
  input x;
  if 1<=x<=5 then go to add;
  put x;
  add: sumx+x;
  datalines;
7
6
323
;
```

各 DATA ステップの最後に暗示的な RETURN ステートメントが含まれているため、合計ステートメントを実行すると、プログラムの実行は DATA ステップの先頭に戻されます。そのため、DATA ステップの最後に明示的な RETURN ステートメントを指定する必要はありません。

- 条件が偽になるオブザベーションに対して合計ステートメントを実行しない場合は、コードを変更し、次のように明示的な RETURN ステートメントを指定します。

```

data info;
  input x;
  if 1<=x<=5 then go to add;
  put x=;
  return;
  /* SUM statement not executed */
  /* if x<1 or x>5      */
add: sumx+x;
datalines;
7
6
323
;

```

## 関連項目:

### ステートメント:

- [“DO ステートメント” \(62 ページ\)](#)
- [“ラベル: ステートメント” \(219 ページ\)](#)
- [“LINK ステートメント” \(266 ページ\)](#)
- [“RETURN ステートメント” \(365 ページ\)](#)

---

## IF ステートメント、サブセット化

指定した式の条件に一致するオブザベーションの処理を継続します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション

CAS

**種類:** 実行

## 構文

**IF** *expression*;

## 引数

***expression***

SAS 式を指定します。

## 詳細

### 基本

サブセット化 IF ステートメントにより、DATA ステップにおいて、この IF ステートメントに指定した式の条件を満たす SAS データセットの生データレコードやオブザベーションのみ処理が継続されます。式がオブザベーションやレコードに対して真として評価される場合(値がゼロでも欠損値でもない場合)、DATA ステップ内のステートメントが実行され、現在のオブザベーションがデータセッ

トに挿入されます。実行後に生成される SAS データセットには、オリジナルの外部ファイルまたは SAS データセットのサブセットが含まれます。

式が偽として評価される場合(値が 0 または欠損値である場合)、それ以降のステートメントはオブザベーションやレコードに対して実行されません。現在のオブザベーションはデータセットに書き込まれず、DATA ステップの残りのプログラムステートメントは実行されません。サブセット化 IF ステートメントでは、オブザベーションの処理を中止するために他のステートメントを実行する必要がないため、すぐに DATA ステップの先頭に戻ります。

### CONTAINS 演算子や LIKE 演算子に相当する演算子を IF ステートメントで使用する

WHERE 句内の LIKE 演算子は、複数のワードからなるパターンと一致します。IF ステートメントで同じ結果を得るには、'='演算子を使用します。この演算子は、文字列の先頭から始まるパターンと一致します。たとえば次のように記述します。

```
data test;
  input name $;
  datalines;
John
Diana
Diane
Sally
Doug
David
;
run;
```

```
data test;
  set test;
  if name = 'D';
run;
```

```
proc print;
run;
```

WHERE 句内の CONTAINS 演算子は、値の内部に指定の文字列が含まれているかどうかをチェックします。IF ステートメントで同じ結果を得るには、INDEX 関数を使用します。たとえば次のように記述します。

```
data test;
  set test;
  if index(name,'ian') ge 1;
run;
```

```
proc print;
run;
```

### 比較

- サブセット化 IF ステートメントは、次の IF-THEN ステートメントと同じです。

```
if not (expression)
  then delete;
```

- SAS データセットの作成時、オブザベーションを含める条件を指定するほうが簡単な場合はサブセット化 IF ステートメントを使用します。オブザベーションを除外する条件を指定するほうが簡単な場合は DELETE ステートメントを使用します。
- サブセット化 IF ステートメントと WHERE ステートメントは同じではありません。この 2 つのステートメントでは、動作や生成される出力データセットが異なる場合があります。重要な違いは次の点になります。
  - サブセット化 IF ステートメントでは、プログラムデータベクトルに読み込まれたオブザベーションを選択します。WHERE ステートメントでは、プログラムデータベクトルに読み込まれる前にオブザベーションを選択します。サブセット化 IF ステートメントでは、入力データセットの各オブザベーションをプログラムデータベクトルに読み込む必要があるため、WHERE ステートメントと比較すると効率的ではない場合があります。
  - SAS データセットのインタリーブ、マージ、更新を実行する DATA ステップでは、サブセット化 IF ステートメントと WHERE ステートメントの結果が異なる場合があります。
  - サブセット化 IF ステートメントと MERGE ステートメントを同時に使用する場合、現在のオブザベーションを結合した後にオブザベーションを選択します。WHERE ステートメントと MERGE ステートメントを同時に使用する場合、現在のオブザベーションを結合する前に各データセットに選択基準を適用します。
  - サブセット化 IF ステートメントでは、既存の SAS データセットや、INPUT ステートメントを使用して読み込んだ生データからオブザベーションを選択できます。WHERE ステートメントでは、既存の SAS データセットからのみオブザベーションを選択できます。
  - サブセット化 IF ステートメントは実行ステートメントですが、WHERE ステートメントは実行ステートメントではありません。

### 例: オブザベーションの限定

- 次の例では、変数 SEX の値が F であるオブザベーションのみがデータセットに含まれます。

```
if sex='F';
```

- 次の例では、変数 AGE の値が欠損値でも 0 でもないすべてのオブザベーションがデータセットに含まれます。

```
if age;
```

### 関連項目:

#### データセットオプション:

- [“WHERE= Data Set Option” \(SAS Viya Data Set Options: Reference\)](#)

#### ステートメント:

- [“DELETE ステートメント” \(60 ページ\)](#)
- [“IF-THEN/ELSE ステートメント” \(137 ページ\)](#)
- [“WHERE ステートメント” \(416 ページ\)](#)

---

## IF-THEN/ELSE ステートメント

特定の条件を満たすオブザベーションに対して、SAS ステートメントを実行します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 実行

---

### 構文

```
IF expression THEN statement;  
<ELSE statement;>
```

### 引数

#### *expression*

任意の SAS 式を指定します。この引数は必須です。

#### *statement*

実行 SAS ステートメントまたは DO グループを指定できます。

### 詳細

SAS では、IF-THEN ステートメントの式を評価し、非ゼロ、ゼロ、または欠損のいずれかの結果を生成します。評価結果が非ゼロおよび非欠損の場合、この式は真になります。評価結果がゼロまたは欠損の場合、この式は偽になります。

IF 句に指定した条件に一致すると、IF-THEN ステートメントは SAS データセットから読み込んだオブザベーション、外部ファイルのレコード、または計算値に対して、SAS ステートメントを実行します。THEN 句が実行されない場合、オブザベーションの ELSE ステートメントが代替アクションを指示します。ELSE ステートメントを使用する場合、IF-THEN ステートメントの後に指定します。

ELSE ステートメントを *指定せず* に IF-THEN ステートメントを使用する場合、すべての IF-THEN ステートメントが評価されます。ELSE ステートメントを *指定して* IF-THEN ステートメントを使用する場合、真のステートメントが検出されるまで IF-THEN ステートメントが実行されます。その後の IF-THEN ステートメントは評価されません。

**注:** 効率を高めるには、IF-THEN/ELSE ステートメントに条件をその確率の降順になるように指定します。

### 比較

- 相互排他な多数の条件を指定する場合、IF-THEN ステートメントではなく SELECT グループを使用します。
- IF 句に指定した条件に一致するオブザベーションまたはレコードのみの処理を続けるには、THEN 句を指定せずにサブセット化 IF ステートメントを使用します。

## 例: IF-THEN/ELSE ステートメントの各種の指定方法

これらの例では、IF-THEN/ELSE ステートメントの各種の指定方法を示します。

- `if x then delete;`
- `if status='OK' and type=3 then count+1;`
- `if age ne agecheck then delete;`
- `if x=0 then`  
`if y ne 0 then put 'X ZERO, Y NONZERO';`  
`else put 'X ZERO, Y ZERO';`  
`else put 'X NONZERO';`
- `if answer=9 then`  
`do;`  
`answer=.`;  
`put 'INVALID ANSWER FOR ' id=;`  
`end;`  
`else`  
`do;`  
`answer=answer10;`  
`valid+1;`  
`end;`
- `data region;`  
`input city $ 1-30;`  
`if city='New York City'`  
`or city='Miami' then`  
`region='ATLANTIC COAST';`  
`else if city='San Francisco'`  
`or city='Los Angeles' then`  
`region='PACIFIC COAST';`  
`datalines;`  
`...more data lines...`  
`;`

### 関連項目:

#### ステートメント:

- [“DO ステートメント” \(62 ページ\)](#)
- [“IF ステートメント、サブセット化” \(134 ページ\)](#)
- [“SELECT ステートメント” \(375 ページ\)](#)

---

## %INCLUDE ステートメント

SAS プログラミングステートメント、データ行、またはその両方を現在の SAS プログラムに読み込みます。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**別名:** %INC

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

## 構文

```
%INCLUDE source(s) </<SOURCE2> <S2=length>
<Linux-options> >;
```

## 引数

### source(s)

%INCLUDE ステートメントでアクセスする情報の格納場所を指定します。ソースとして次の3つが考えられます。

ソース	定義
file-specification	外部ファイルを指定します。
internal-lines	同一の SAS ジョブまたは SAS セッションで入力した行を指定します。
keyboard-entry	キーボードから直接入力するステートメントまたはデータ行を指定します。

### file-specification

プログラムに読み込む外部ファイルを指定します。

*File-specification* には、次の形式を指定できます。

#### 'external-file'

一重引用符で囲んだ外部ファイルの物理名を指定します。物理名には動作環境でファイルを判別できる名前を指定します。

### fileref

外部ファイルに関連付けられているファイル参照名を指定します。

ヒント FILENAME ステートメント、FILENAME 関数、または動作環境のコマンドを使用すると、ファイル参照名の関連付けを行うことができます。

### fileref (filename-1 <, "filename-2.xxx", ... filename-n>)

集約記憶域に関連付けられているファイル参照名を指定します。ファイル参照名の後ろに、指定した場所に存在するファイル名を1つまたは複数指定します。また、各ファイルはカンマまたはブランクで区切ります。次に指定したファイル名全体を丸かっこで囲みます。

この例では、testcode1.sas、testcode2.sas、testcode3.txt を読み込むように指示します。これらのファイルは、集約記憶域 **mylib** に格納されています。testcode1 および testcode2 の拡張子はデフォルトの.SAS なので、ファイル拡張子を指定する必要はありません。testcode3.txt は、.SAS 以外の拡張子なので、指定するファイル名を引用符で囲む必要があります。

```
%include mylib(testcode1, testcode2,
"testcode3.txt");
```

動作環境 動作環境によって、ディレクトリ、MACLIB、区分データセットなど複数のファイルをまとめて保存する場所の名前は異なります。

注	集約記憶域に格納されているファイルの名前が有効な SAS 名ではない場合は、その名前を引用符で囲む必要があります。
ヒント	FILENAME ステートメント、FILENAME 関数、または動作環境のコマンドを使用すると、ファイル参照名の関連付けを行うことができます。
制限事項	外部ファイルから特定の行だけを読み込むことはできません。
動作環境	ファイル名に使用できる文字列の長さは動作環境によって異なります。
ヒント	ERRORCHECK オプションが STRICT に設定されている場合、SYSERR マクロ変数を使用すると、 <i>file-specification</i> の指定の有無を確認できます。
	バッチモード、対話型ラインモード、非対話型モードなど、どの種類の SAS 処理でも、外部ファイルを指定すると便利です。

### *internal-lines*

同一の SAS ジョブまたは SAS セッションで入力した行を指定します。入力済みの行を読み込むには、次のいずれかの形式を使用します。

*n*                                    *n* 番目の行を読み込みます。

*n-m* または *n:m*    *n* 番目から *m* 番目までの行を読み込みます。

注 対話型ラインモード、非対話型モード、バッチモードを実行している場合は、SPOOL システムオプションによってサブミット済みの行への内部アクセスが制御されます。デフォルトでは、SPOOL システムオプションは NOSPOOL に設定されます。%%INCLUDE ステートメントを使用して入力済みの行を参照するには、SPOOL システムオプションを有効にする必要があります。システム上での SPOOL システムオプションの現在の設定を確認するには、OPTIONS プロシジャを使用します。

ヒント 入力済みの行の読み込みは、対話型ラインモードの処理で使用すると便利です。

読み込む行番号を指定するには、%LIST ステートメントを使用します。

### *keyboard-entry*

プログラムを待機させる方法の 1 つです。この方法を使用すると、現在の実行中のプログラムの処理を中断し、キーボードからステートメントやデータ行を入力した後、プログラムの処理を再開できます。

\*

キーボードからデータを入力するように求めるプロンプトを表示します。コード内で%INCLUDE ステートメントの後ろにアスタリスク(\*)を指定します。

```
proc print;
  %include *;
run;
```



元のソースプログラムの処理を再開するには、キーボードから%RUN ステートメントを入力します。

注 FILENAME ステートメント、FILENAME 関数、または動作環境のコマンドを使用して、ファイル参照名 SASTERM を事前に外部ファイルに関連付けておく必要があります。

ヒント SAS を非対話型モードまたは対話型ラインモードで実行する場合は、この方法を使用します。SAS は処理を停止し、キーボードからステートメントを入力するように求めるプロンプトを表示します。

キーボードからソースを読み込むにはこの引数を指定します。

%INCLUDE \* ステートメントをバッチジョブでも使用できます。その場合、キーボードから入力するかわりに、ステートメントが含まれるファイルを作成して、ファイル参照名 SASTERM を関連付けます。%%INCLUDE \* ステートメントを実行すると、SASTERM を割り当てたファイルからステートメントを読み込みます。SASTERM を割り当てたファイルでは、元のソースから読み込みを再開する位置に%RUN ステートメントを追加します。

## SOURCE2

SAS プログラムに読み込むソースステートメントを SAS ログに表示します。

ヒント SAS ログには、ファイル参照名、ソースのファイル名、ネストレベル (1、2、3 など) も表示されます。

SOURCE2 システムオプションでも、結果は同じになります。%INCLUDE ステートメントに SOURCE2 を指定すると、読み込み処理中は、SOURCE2 システムオプションの設定より優先されます。

## S2=length

入力に使用するレコード長を指定します。length には、次の値を指定できません。

- S SAS システムオプション S=の現在の設定と等しい値に S2 を設定します。
- 0 SEQ=システムオプションの設定を使用して、行にシーケンスフィールドが含まれているかどうかを判定するよう指定します。行にシーケンスフィールドが含まれている場合、SAS システムは、そのシーケンスフィールドを合計長から除外することにより行の長さを決定します。
- n ファイルが固定長レコードを含んでいる場合に、読み込み対象とする行の長さに対応するゼロより大きい数を指定します。ファイルが可変長レコードを含んでいる場合、n は、データの読み込みを開始するカラムを指定します。

操作 S2=システムオプションでは、%INCLUDE ステートメントでアクセスするセカンダリソースステートメントの長さも指定します。この値は、SAS セッション実行中は有効です。%%INCLUDE に指定した S2=オプションは、現在の読み込み処理でのみ有効です。%%INCLUDE ステートメントに S2=オプションを指定すると、読み込み処理中は、システムオプションの設定より優先されます。

ヒント %INCLUDE ステートメントから読み込むテキストは、固定長または可変長のどちらかになります。

固定長レコードは、シーケンスフィールドなしレコードか、またはレコードの末尾にシーケンスフィールドを有するレコードです。固定長レコードの場合、S2=オプションに指定した値は、データの末尾の列位置を示します。

可変長レコードは、シーケンスフィールドなしレコードか、またはレコードの先頭にシーケンスフィールドを有するレコードです。固定長レコードの場合、S2=オプションに指定した値は、データの先頭の列位置を示します。

参照項目: 固定長入力レコードおよび可変長入力レコードの詳細については、“[S=System Option](#)” (*SAS Viya System Options: Reference*)および“[S2=System Option](#)” (*SAS Viya System Options: Reference*)を参照してください。

## Linux 引数

### ENCODING=*encoding-value*'

指定されたソースから読み込むときに使用するエンコーディングを指定します。ENCODING=の値は、指定されたソースのエンコーディングが、現在のセッションのエンコーディングとは異なることを示します。

指定されたソースのデータを読み込むときは、指定されたエンコーディングからセッションのエンコーディングへ、SAS によってデータがトランスコードされます。

エンコーディングの有効な値については、“[Overview to SAS Language Elements That Use Encoding Values](#)” (*SAS Viya National Language Support: Reference Guide*)を参照してください。

制限事項 ENCODING=オプションは、予約済みのファイル参照名ではないファイル指定が INFILE ステートメントに含まれている場合のみ有効です。INFILE ステートメントの ENCODING=値は、ENCODING=システムオプションの値をオーバーライドします。

### BLKSIZE=*block-size*

#### BLK=*block-size*

1 回の I/O 操作で物理的に書き込まれたり読み込まれたりするバイト数を指定します。デフォルト値は 8K です。最大値は 1M です。

### LRECL=*record-length*

論理レコード長(バイト単位)を指定します。LRECL=のデフォルト値は 32,767 です。固定長レコード(RECFM=F)を使用する場合、LRECL=のデフォルト値は 256 になります。*record-length* 値の可能な範囲は、1 から 1,048,576 (1 MB)です。

### RECFM=*record-format*

レコード形式を指定します。Linux 環境では次の値が使用できます。

- D デフォルトのレコード形式(可変長形式)。
- F 固定長形式。つまり、各レコードの長さは同じです。
- N バイナリ形式。ファイルはバイトストリームで構成され、レコード境界はありません。

- P            プリント形式。
- V            可変長形式。各レコードは改行文字で終わります。
- S370V       可変長 S370 レコード形式(V)。
- S370VB      可変長ブロック S370 レコード形式(VB)。
- S370VBS    スパンレコード可変長ブロック S370 レコード形式(VBS)。
- S370 値は、z/OS ファイルとして配置されたファイルの場合のみ有効です。つまり、バイナリで、variable-length レコードを持ち、EBCDIC 形式であるファイルです。固定長形式の z/OS ファイルを使用する場合は、まずそれを variable-length のバイナリの z/OS ファイルにコピーします。

## 詳細

### %INCLUDE の動作

%INCLUDE ステートメントが含まれるプログラムを実行すると、記述したコードが実行されます。このコードには、%INCLUDE ステートメントを使用してプログラムに読み込むステートメントやデータ行が含まれます。

#### 動作環境の情報

%INCLUDE ステートメントの使用方法は、操作環境によって異なります。その他のソフトウェアの機能、ファイルの参照方法やファイルへのアクセス方式の詳細については、お使いの動作環境向けのドキュメントを参照してください。また、%INCLUDE ステートメントの例を実行する前にも、それらのドキュメントを参照してください。

### 3つのデータソース

%INCLUDE ステートメントは、次の3つのソースの SAS ステートメントやデータ行にアクセスします。

- 外部ファイル
- 同一のジョブやセッションで入力した行
- キーボードから入力した行

### 有用な場合

%INCLUDE ステートメントは、対話型ラインモード、非対話型モード、またはバッチモードで SAS を実行する場合によく使用されます。SAS Studio を使用して SAS を実行する場合にも%INCLUDE ステートメントを使用できますが、INCLUDE コマンドと RECALL コマンドを使用してデータ行やプログラムステートメントにアクセスし、それらの行を再びサブミットを実行する方がより実用的です。

### %INCLUDE 使用のルール

- ステートメントには、ソースの数をいくつでも指定できるほか、読み込むソースの種類を混在させることもできます。1つの%INCLUDE ステートメントでは複数のソースから情報を読み込むことができますが、ソースごとに%INCLUDE ステートメントを使用したほうがプログラムがわかりやすくなります。
- %INCLUDE ステートメントは、ステートメント境界から記述を開始する必要があります。つまり、SAS ジョブの最初のステートメントとするか、他のステートメントの末尾のセミコロンの直後に記述する必要があります。

%INCLUDE ステートメントでアクセスするファイルの中に別の%INCLUDE ステートメントがネストされていてもかまいません。%INCLUDE ステートメントで指定できる最大ネスト数は、動作環境に固有の制限事項(メモリ容量や同時に開くことができるファイルの最大数など)によって異なります。

- %INCLUDE ステートメントはグローバルステートメントですが、グローバルステートメントは実行可能ではありません。そのため、%INCLUDE ステートメントは条件付きロジックでは使用できません。
- 行の最大長は 32,000 バイトです。

## 比較

%INCLUDE ステートメントでは、ステートメントを直ちに実行します。INCLUDE コマンドは、読み込んだ行を SAS Studio の **Code** ウィンドウに表示しますが、それらを実行しません。表示された行を実行するには、SUBMIT コマンドを発行する必要があります。

## 例

### 例 1: 外部ファイルの挿入

- この例では、プログラムの一部をファイルに格納し、後からプログラムに読み込みます。このプログラムは、MYFILE という名前のファイルに格納されています。

```
data monthly;
  input x y month $;
  datalines;
1 1 January
2 2 February
3 3 March
4 4 April
;
```

次のプログラムでは、外部ファイル MYFILE を読み込み、そのファイルに含まれる DATA ステップをサブミットしてから、PROC PRINT ステップを実行します。

```
%include 'MYFILE';
proc print;
run;
```

- 実際のファイル名かわりにファイル参照名を使用してファイルを参照する場合、FILENAM ステートメント(または動作環境コマンド)を使用してファイル参照名を割り当てることができます。

```
filename in1 'MYFILE';
```

これ以後は、ファイル参照名 IN1 を使用して MYFILE にアクセスできます。

```
%inc in1;
```

- ディレクトリ、PDS、MACLIB(または動作環境で集約記憶域を指す名称)に格納されたファイルを使用する場合、サイズが大きいストレージユニットにファイル参照名を割り当ててから、ファイル名を指定します。たとえば、次の FILENAME ステートメントでは、ファイル参照名 STORAGE を集約記憶域に割り当てます。

```
filename storage
  'aggregate-storage-location';
```

これ以降は、次のステートメントを使用してファイルを読み込むことができます。

```
%inc storage(MYFILE);
```

- 1つの%INCLUDE ステートメントでファイル参照名を指定した後に、複数のファイルやメンバを丸かっこで囲んで指定すると、この記憶域のファイルやメンバにアクセスすることもできます。ファイル名はカンマやブランクで区切ります。次の%INCLUDE ステートメントは、この方法を示しています。

```
%inc storage(file-1,file-2,file-3);
```

- デフォルトの.sas 拡張子ではないファイルを指定する場合、各ファイル名を引用符で囲んでから、指定するファイル名全体を丸かっこで囲むとファイルにアクセスできます。

```
%inc storage("file-1.txt","file-2.dat",
"file-3.cat");
```

### 例 2: 事前にサブミットされた行を挿入する

次の%INCLUDE ステートメントでは、行 1、行 5、行 9 - 12、行 13 - 16 をキーボードから再入力したのと同じように実行します。

```
%include 1 5 9-12 13:16;
```

### 例 3: キーボード入力を含める

この例に示す方法は、非対話型モードまたは対話型ラインモードを使用している場合にだけ使用できます。

この例では、%INCLUDE ステートメントを使用して、PROC PRINT の実行時にカスタマイズした TITLE ステートメントを追加します。

```
data report;
  infile file-specification;
  input month $ salesamt $;
run;
proc print;
  %include *;
run;
```

この DATA ステップを実行すると、%INCLUDE ステートメントにアスタリスクが指定されているので、キーボードからステートメントを入力するように求めるプロンプトが表示されます。次のようなステートメントを入力できます。

```
where month= 'January';
title 'Data for month of January';
```

ステートメントを入力した後、次のように入力することで、%RUN ステートメントを使用して処理を再開できます。%run;

%RUN ステートメントは、キーボード入力モードの終了を知らせ、元のプログラムの残りのステートメントの実行を再開するように命令します。

### 例 4: %INCLUDE を使用して 1 つのカタログから複数のエントリを読み込む

この例では、カタログ MYLIB.INCLUDE にある 3 つのエントリからソースコードをサブミットします。エントリタイプを指定しない場合、デフォルトのエントリタイプ CATAMS に設定されます。

```
filename dir catalog 'mylib.include';
%include dir(mem1);
```

```
%include dir(mem2);
%include dir(mem3);
```

## 関連項目:

### ステートメント:

- “%LIST ステートメント” (270 ページ)
- “%RUN ステートメント” (368 ページ)

---

## INFILE ステートメント

INPUT ステートメントで読み込む外部ファイルを指定します。

**該当要素:** DATA ステップ

**カテゴリ:** ファイル操作

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。  
SAS がロックダウン状態にある場合、ロックダウンパスリストに含まれていないファイルに関しては、INFILE ステートメントを使用できません。

## 構文

```
INFILE file-specification <device-type> <option(s)> ;
```

## 引数

### *file-specification*

入力データレコードのソースを指定します。これは外部ファイルまたは入力ストリームデータになります。*File-specification* には、次の形式を指定できません。

#### 'external-file'

外部ファイルの物理名を指定します。物理名とは、動作環境がそのファイルにアクセスするために使用する名前のことです。

#### *fileref*

外部ファイルのファイル参照名を指定します。

**要件** ファイル参照名は、FILENAME ステートメントまたは FILENAME 関数を使用するか、適切な動作環境のコマンドを使用して、あらかじめ前のステップで外部ファイルに関連付けておく必要があります。

**参照項目:** “FILENAME ステートメント” (96 ページ)

#### *fileref(file)*

集約記憶域に関連付けられているファイル参照名の後に、その集約記憶域に含まれているファイルまたはメンバの名前を丸かっこで囲んで指定します。

**要件** 集約記憶域に格納されているファイルの名前が有効な SAS 名ではない場合は、その名前を引用符で囲む必要があります。

ファイル参照名は、FILENAME ステートメントまたは FILENAME 関数を使用するか、適切な動作環境のコマンドを使用して、あらかじめ前のステップで外部ファイルに関連付けておく必要があります。

**参照項目:** [“FILENAME ステートメント” \(96 ページ\)](#)

#### CARDS | CARDS4

定義については、[DATALINES \(147 ページ\)](#) を参照してください。

**別名** DATALINES | DATALINES4

#### DATALINES | DATALINES4

DATA ステップ内の DATALINES または DATALINES4 ステートメントの直後に入力データが続くことを指定します。DATALINES を指定すると、INFILE ステートメントの各種オプションを使用して、INPUT ステートメントが入力ストリームデータ行を読み込む方法を制御できます。

**別名** CARDS | CARDS4

**例** [“例 1: 区切り文字の扱い方の変更” \(167 ページ\)](#)

**ヒント** ERRORCHECK オプションが STRICT に設定されている場合、SYSERR マクロ変数を使用すると、*file-specification* の指定の有無を確認できます。

#### **device-type**

ファイル参照名が入力デバイスまたは出力デバイスを参照する場合や物理ファイルではない場所を参照する場合に使用する、デバイスタイプまたはアクセス方式を指定します。

#### **ACTIVEMQ**

このアクセス方式を指定すると、ActiveMQ メッセージブローカーへアクセスできます。

**操作** DATA ステップでアクセス方式のオプションを認識できない場合、DATA ステップはこのオプションをアクセス方式に渡して処理します。

#### **DISK**

デバイスがディスクドライブであると指定します。

**ヒント** ディスク上のファイルにファイル参照名を割り当てる場合、DISK を指定する必要はありません。

#### **DUMMY**

ファイルへの出力を破棄するように指定します。

**ヒント** テストを実行する場合は DUMMY を指定すると便利です。

#### **GTERM**

出力デバイスの種類がグラフィックデータを受信するグラフィックデバイスであると指定します。

**JMS**

Java Message Service (JMS) の送信先を指定します。

**PIPE**

名前の付いていないパイプを指定します。

注 動作環境によっては、パイプがサポートされない場合があります。

**PLOTTER**

バッファなしのグラフィック出力デバイスを指定します。

**PRINTER**

プリンタまたはプリンタスプールファイルを指定します。

**SFTP**

SFTP アクセス方式を指定します。

**操作** DATA ステップでアクセス方式のオプションを認識できない場合、DATA ステップはこのオプションをアクセス方式に渡して処理します。

**参照項目:** SFTP アクセス方式で指定可能なオプションの一覧については、“[FILENAME ステートメント、SFTP アクセス方式](#)” (109 ページ)を参照してください。

**TAPE**

テープドライブを指定します。

**TEMP**

ファイル名が割り当てられている間だけ存在する一時ファイルを作成します。この一時ファイルは論理名からのみアクセスできます。また、論理名が存在する間だけ使用できます。

**制限事項** 物理パス名は指定しないでください。物理パス名を指定するとエラーが発生します。

**ヒント** TEMP デバイスで操作されるファイルは、同じ属性を持ち、DISK ファイルと同様な動作ができます。

**TERMINAL**

ユーザーの端末を指定します。

**UPRINTER**

ユニバーサル印刷プリンタの定義名を指定します。

**ヒント** FILENAME ステートメントにプリンタ名を指定しない場合、PRINTERPATH オプションによって、使用するユニバーサルプリンタと出力先が制御されます。

**URL**

URL アクセス方式を指定します。

**操作** DATA ステップでアクセス方式のオプションを認識できない場合、DATA ステップはこのオプションをアクセス方式に渡して処理します。

**参照項目:** URL アクセス方式で指定可能なオプションの一覧については、“[FILENAME ステートメント、URL アクセス方式](#)” (115 ページ)を参照してください。



別名	DEVICE= <i>device-type</i>
デフォルト	DISK
要件	<i>device-type</i> または DEVICE= <i>device-type</i> は、ステートメントの <i>file-specification</i> の直後に記述する必要があります。

## INFILE オプション

### BLKSIZE=*block-size*

入力ファイルのブロックサイズをバイトで指定します。

デフォルト 8K.最大は 1G-1 です。

### COLUMN=*variable*

変数名を指定します。指定した変数の値は、入力ポインタの現在の列位置を示す値に設定されます。自動変数と同じように、COLUMN=に指定した変数はデータセットに書き込まれません。

別名 COL=

参照項目: [LINE= \(154 ページ\)](#)

例 “例 8: ポインタ位置の出力” (172 ページ)

### DELIMITER= *delimiter(s)*

LIST 入力で空白のかわりに使用する区切り文字を指定します。使用できる *delimiter(s)*は次のとおりです。

#### '*list-of-delimiting-characters*'

区切り文字として読み込む文字を 1 つまたは複数指定します。

要件 指定する区切り文字のリストは引用符で囲む必要があります。

例 “例 1: 区切り文字の扱い方の変更” (167 ページ)

#### *character-variable*

文字変数の名前を指定します。指定した変数の値が区切り文字として使用されます。

別名 DLM=

デフォルト ブランク

ヒント 区切り文字では大文字と小文字が区別されます。

一般的な区切り文字としては、カンマ(,)、縦棒(|)、セミコロン(;)、タブが挙げられます。タブは 16 進数の文字で指定します。

Linux 環境では、タブの値は'09'x になります。

参照項目: “区切り文字で区切られたデータ値の読み込み” (161 ページ), DLMSTR= (150 ページ)、および “DSD (delimiter-sensitive data)” (151 ページ)

例 “例 1: 区切り文字の扱い方の変更” (167 ページ)

**DLMSTR= delimiter**

LIST 入力で空白のかわりに使用する区切り文字を指定します。使用できる *delimiter* は次のとおりです。

**'delimiting-string'**

区切り文字として読み込む文字列を指定します。

要件 文字列は一重引用符で囲みます。

例 “例 1: 区切り文字の扱い方の変更” (167 ページ)

**character-variable**

文字変数の名前を指定します。指定した変数の値が区切り文字として使用されます。

デフ    ブランク  
オル  
ト

操作    INFILE ステートメントに DLMSTR=オプションを複数指定すると、最後に指定した DLMSTR=オプションが使用されます。DELIMITER=オプションと DLMSTR=オプションの両方を指定すると、最後に指定したオプションが使用されます。

RECFM=N を指定する場合は、サイズが大きい入力項目でも十分に保持できる値が LRECL に指定されていることを確認してください。指定した値が十分ではない場合、区切り文字によってレコード境界間で分割される場合があります。

ヒント    区切り文字では大文字と小文字が区別されます。区切り文字で大文字小文字を区別するには、DLMSOPT='I'オプションを使用します。

参照項目: “区切り文字で区切られたデータ値の読み込み” (161 ページ)、  
DELIMITER= (149 ページ)、DLMSOPT= (150 ページ)、および DSD (151 ページ)

例        “例 1: 区切り文字の扱い方の変更” (167 ページ)

**DLMSOPT= 'option(s)'**

DLMSTR=オプションの解析オプションを指定します。ここでは、*option(s)*は次の値のいずれかになります。

I

大文字小文字を区別せずに比較を行うことを指定します。

T

区切り文字の末尾にある空白を削除するよう指定します。

ヒント    Tオプションは、区切り文字として変数を使用する場合に便利です。

I、T、またはその両方を指定できます。

要件        DLMSOPT=オプションは、DLMSTR=オプションを使用している場合にのみ有効です。

参照項目: DLMSTR= (150 ページ)

## 例 “例 1: 区切り文字の扱い方の変更” (167 ページ)

**DSD (delimiter-sensitive data)**

データ値が引用符で囲まれている場合、そのデータ値内の区切り文字を文字データとして扱うことを指定します。DSD オプションを指定すると、LIST 入力を使用する場合の SAS による区切り文字の扱い方を変更し、デフォルトの区切り文字をカンマに変更できます。DSD を指定すると、SAS は 2 つの連続する区切り文字を欠損値として扱い、文字値から引用符を削除します。

**操作** 区切り文字を変更するには、DELIMITER=または DLMSTR=オプションを使用します。

**ヒント** 引用符に囲まれている文字列内に区切り文字を含んでいる文字値を読み込むには、DSD オプションと LIST 入力を使用します。INPUT ステートメントは、そのような区切り文字を有効な文字として扱い、その値を格納する前に、文字列から引用符を削除します。引用符を保持するには、チルダ(~)フォーマット修飾子を使用します。

**参照項目:** “区切り文字で区切られたデータ値の読み込み” (161 ページ)、DELIMITER= (149 ページ)、および DLMSTR= (150 ページ)

例 “例 1: 区切り文字の扱い方の変更” (167 ページ)

“例 2: 欠損値と短いレコードをリスト入力で処理する” (169 ページ)

**ENCODING= 'encoding-value'**

外部ファイルからの読み取り時に使用するエンコーディングを指定します。ENCODING=の値は、外部ファイルのエンコーディングが現在のセッションエンコーディングとは異なることを示しています。

外部ファイルからデータを読み込む場合は、指定したエンコーディングからセッションエンコーディングにデータがトランスコードされます。

**デフォルト** SAS では、外部ファイルのエンコーディングがセッションエンコーディングと同じであるとみなします。

**参照項目:** エンコーディングの有効な値については、“Encoding Values in SAS Language Elements” (*SAS Viya National Language Support: Reference Guide*)を参照してください。

例 “例 11: 外部ファイル読み込み時のエンコードの指定” (175 ページ)

**END=variable**

変数名を指定します。現在の入力データレコードが最後である場合、この変数に値 1 が設定されます。SAS が最後のデータレコードを処理するまで、END=変数には値 0 が設定されます。自動変数と同じように、この変数はデータセットに書き込まれません。

**制限事項** END=オプションは、UNBUFFERED オプションと一緒に使用できません。また、複数の入力データレコードを読み込む DATALINES/DATALINES4 ステートメントや INPUT ステートメントでは使用できません。

**ヒント** END=が無効である場合、オプション EOF= (152 ページ) を使用します。

例 “例 5: 複数の入力ファイルからの読み込み” (171 ページ)

### EOF=label

ステートメントラベルを指定します。これは、INFILE ステートメントがファイルの終わりに達した場合に明示的な GO TO のオブジェクトとなります。INPUT ステートメントが、それ以上レコードを含んでいないファイルから読み込もうとする場合、このステートメントラベルに実行が移動されます。

**操作** UNBUFFERED オプションと一緒に使用する場合や、複数の入力データレコードを読み込む DATALINES/DATALINES4 ステートメントや INPUT ステートメントと組み合わせて使用する場合、END=オプションのかわりに EOF=を使用します。

**ヒント** EOF=オプションは、複数の入力ファイルを順番に読み込む場合に便利です。

**参照項目:** [END= \(151 ページ\)](#)、[EOV= \(152 ページ\)](#)、および [UNBUFFERED \(159 ページ\)](#)

### EOV=variable

変数名を指定します。一連の連結ファイルに含まれている各ファイルの最初のレコードが読み込まれた場合、この変数に値 1 が設定されます。この変数は、SAS が次のファイルを検出した場合にのみ値が設定されます。自動変数と同じように、EOV=に指定した変数はデータセットに書き込まれません。

**ヒント** SAS が各ファイル境界を検出した時点で、EOV=変数は 0 にリセットされます。

**参照項目:** [END= \(151 ページ\)](#) および [EOF= \(152 ページ\)](#)

### EXPANDTABS | NOEXPANDTABS

タブ文字を標準タブ設定に拡張するかどうかを指定します。EXPANDTABS を指定すると、タブ文字が、列 9 から始まる 8 列の間隔に設定されます。

**デフォルト** NOEXPANDTABS

**ヒント** EXPANDTABS オプションは、お使いの動作環境にネイティブなタブ文字を含むデータを読み込む場合に便利です。

### FILENAME=variable

変数名を指定します。この変数には、現在開かれている入力ファイルの物理名が設定されます。一連の連結ファイルでは、この変数は、SAS が次のファイルを検出した場合にのみ更新されます。自動変数と同じように、FILENAME=に指定した変数はデータセットに書き込まれません。

**ヒント** LENGTH ステートメントを使用すると、物理ファイル名を格納できる長さにまで変数の長さを拡張できます。

**参照項目:** [FILEVAR= \(152 ページ\)](#)

例 “例 5: 複数の入力ファイルからの読み込み” (171 ページ)

### FILEVAR=variable

変数名を指定します。この変数の値が変化すると、INFILE ステートメントは現在の入力ファイルを閉じ、新しいファイルを開きます。次の INPUT ステートメントが実行されると、FILEVAR=変数に指定された新しいファイルから読

み込みが実施されます。自動変数と同じように、この変数はデータセットに書き込まれません。

**制限事項** FILEVAR=変数には、物理ファイル名を表す文字列を指定する必要があります。

**操作** FILEVAR=オプションを使用すると、*file-specification* は、実際のファイル名またはファイルに対して事前に割り当てられたファイル参照名ではなく、プレースホルダになります。このプレースホルダを使用して、処理情報を SAS ログに出力します。プレースホルダには、ファイル参照名と同じルールが適用されます。

**ヒント** FILEVAR=オプションを使用すると、現在開かれている入力ファイルを新しい物理ファイルに動的に変更できます。

FILEVAR=オプションを使用する場合、現在開かれている入力ファイルが最後のファイルであるかどうかを知ることはできません。DATA ステップがファイル終端マーカまたはすべての開かれているデータセットの終わりに到達すると、シャットダウンが行われます。また、FILEVAR を FIRSTOBS と併用すると、複数のファイルの中でヘッダーレコードのみを含むファイルによって、DATA ステップの通常のシャットダウンがトリガされます。このシャットダウンは、ファイル終端指示子を超えて読み込みを行うと DATA ステップが終了するために発生します。EOF=オプションを使用すると、シャットダウンを回避することができます。

**参照項目:** “外部ファイルの直接更新” (160 ページ)

**例** “例 5: 複数の入力ファイルからの読み込み” (171 ページ)

### **FIRSTOBS=record-number**

レコード番号を指定します。SAS はこの番号を使用して、入力ファイル内の入力データレコードの読み込みを開始します。

**デフォルト** 1

**ヒント** FIRSTOBS=と OBS=を組み合わせることで、ファイル内に含まれている特定の範囲のレコードを読み込むことができます。

ファイル内のヘッダーレコードをスキップするには、FIRSTOBS=2 を使用します。

**例** 次のステートメントでは、50 番目から 100 番目までのレコードを処理します。

```
infile file-specification firstobs=50 obs=100;
```

### **FLOWOVER**

INPUT ステートメントが、ステートメント内にあるすべての変数に関して、現在の入力行内に値を検出しなかった場合、次の入力データレコードを継続して読み込むようにします。FLOWOVER は、INPUT ステートメントのデフォルトの動作です。

参照項目: “[行の終わりを超えて読み込む](#)” (165 ページ)、[MISSOVER](#) (155 ページ)、[STOPOVER](#) (158 ページ)、および [TRUNCOVER](#) (158 ページ)

### LENGTH=variable

変数名を指定します。この変数には、現在の入力ファイルの長さが設定されます。SAS は、INPUT ステートメントが実行されるまで、この変数に値を割り当てません。自動変数と同じように、LENGTH=変数はデータセットに書き込まれません。

ヒント このオプションは、フィールド幅が異なる場合に、[\\$VARYING 入力形式](#)と組み合わせて使用すると便利です。

例 “[例 4: 可変長レコードを含むファイルの読み込み](#)” (170 ページ)

“[例 7: コピーされたレコードの切り捨て](#)” (172 ページ)

### LINE=variable

変数名を指定します。この変数には、入力バッファ内の入力ポインタの行位置が設定されます。自動変数と同じように、LINE=に指定した変数はデータセットに書き込まれません。

範囲 1 から N=オプションに指定された値

操作 LINE=変数の値は、INPUT ステートメントで N=オプションまたは #n 行ポインタコントロールによって指定された行グループ内の現在の相対行番号です。

参照項目: [COLUMN=](#) (149 ページ)および [N=](#) (155 ページ)

例 “[例 8: ポインタ位置の出力](#)” (172 ページ)

### LINESIZE=line-size

INPUT ステートメントで使用可能なレコード長を指定します。

別名 LS=

範囲 最大 32767

操作 INPUT ステートメントが、LINESIZE=に指定された列数を超えて列を読み込もうとした場合にどのようなアクションが実行されるかは、[FLOWOVER](#)、[MISSOVER](#)、[SCANOVER](#)、[STOPOVER](#)、[TRUNCOVER](#)のうちどのオプションが有効であるかにより決定されます。[FLOWOVER](#)がデフォルトです。

ヒント LINESIZE=を使用すると、レコード全体を読み込みたくない場合にレコード長を制限できます。

例 データ行の 73 列目から 80 列目までに順序番号が含まれている場合、次の INFILE ステートメントを使用すると、INPUT ステートメントによる読み込みを先頭から 72 列目までに制限できます。  

```
infile file-specification linesize=72;
```

### LRECL=logical-record-length

論理レコード長を指定します。この値は、有効なレコード形式によって異なります(RECFM)。



- RECFM=F の場合、LRECL=オプションの値により、1 レコードとして読み取られるバイト数が決定されます。

注: RECFM=F の場合、SAS が SAS9.4 以前のバージョンと通信するには、LRECL=は 256 に設定する必要があります。

- RECFM=N の場合、LRECL=オプションの値は 256 以上にする必要があります。
- RECFM=V の場合、LRECL=オプションの値により最大レコード長が決定されます。指定された長さを超えるレコードは切り捨てられます。

**デフォルト** グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。

**制限事項** DATALINES ファイル指定を使用する場合、LRECL は無効となります。

**操作** かわりに、“LRECL= System Option” ([SAS Viya System Options: Reference](#))を使用すると、グローバルな論理レコード長を指定できます。

**ヒント** LRECL=オプションには、ファイルの物理的な行の長さを指定します。LINESIZE=オプションには、INPUT ステートメントが読み込む行の長さを指定します。

## MISSEVER

INPUT ステートメントが、ステートメント内にあるすべての変数に関して、現在の入力行内に値を検出しなかった場合、新しい入力データレコードを読み込まないようにします。INPUT ステートメントが現在の入力データレコードの終端に到達すると、何の値も割り当てられていない変数には欠損値が設定されます。

**ヒント** 最後のフィールドが欠損値であり、対応する変数に欠損値を割り当てたい場合には、MISSEVER を使用します。

**参照項目:** “[行の終わりを越えて読み込む](#)” (165 ページ)、[FLOWOVER](#) (153 ページ)、[SCANOVER](#) (157 ページ)、[STOPOVER](#) (158 ページ)、および [TRUNCOVER](#) (158 ページ)

**例** “[例 2: 欠損値と短いレコードをリスト入力で処理する](#)” (169 ページ)

## N=available-lines

入力ポインタで一度にアクセスできる行数を指定します。

**デフォルト** DATA ステップ内の任意の INPUT ステートメントでの#ポインタコントロールの後に指定されている最大数。#ポインタコントロールを省略した場合、デフォルト値は 1 になります。

**操作** このオプションは、ポインタで一度にアクセスできる行数にのみ影響します。INPUT ステートメントが読み込む行数には影響しません。

**ヒント** N=の値よりも小さな値の#ポインタコントロールを INPUT ステートメントで使用すると、予期しない結果が発生することがあります。予期しない結果を防ぐには、N=オプションの値に等しい#ポインタコントロールを含めるようにします。たとえば次のように記述します。

```
infile 'external file' n=5;
```

```
input #2 name : $25. #3 job : $25. #5;
```

レコードからデータが読み込まれなくても、INPUT ステートメントには #5 ポインタコントロールが含まれます。

例 “例 8: ポインタ位置の出力” (172 ページ)

### **NBYTE=variable**

変数名を指定します。この変数には、ストリームレコード形式でデータを読み込む場合に(FILENAME ステートメントで RECFM=S を指定)、ファイルから読み込むバイト数が含まれます。

デフォルト ファイルの LRECL 値

### **OBS=record-number | MAX**

*record-number* では、順次読み込みの入力ファイルから読み込む最終レコードのレコード番号を指定します。

MAX では、処理するオブザベーションの最大数を指定します。この値は、少なくとも符号付き 32 ビット整数の最大値と同じ大きさになります。絶対最大値は、お使いの動作環境により異なります。

デフォルト MAX

ヒント OBS=と FIRSTOBS=を組み合わせることで、ファイル内に含まれている特定の範囲のレコードを読み込むことができます。

例 次のステートメントは、ファイルに含まれている先頭から 100 レコードのみを読み込みます。  
infile *file-specification* obs=100;

### **PAD | NOPAD**

LRECL=オプションに指定された長さに達するまで、外部ファイルから読み込むレコードに空白を追加するかどうかを指定します。

デフォルト NOPAD

参照項目: [LRECL=オプション \(154 ページ\)](#)

### **PRINT | NOPRINT**

入力ファイルにキャリッジコントロール文字が含まれているかどうかを指定します。

ヒント キャリッジコントロール文字を削除する必要なしに DATA ステップでファイルを読み込む場合は、PRINT を指定します。キャリッジコントロール文字をデータ値として読み込む場合は、NOPRINT を指定します。

### **RECFM=record-format**

入力ファイルのレコード形式を指定します。*record-format* には、次のいずれかの値を使用できます。

D デフォルトのレコード形式(可変長形式)。

F 固定長形式。つまり、各レコードの長さは同じです。



N	バイナリ形式。ファイルはバイトストリームで構成され、レコード境界はありません。LRECL オプションを指定しない場合、デフォルトではファイルから 256 バイトが読み込まれます。
P	プリント形式。
V	可変長形式。各レコードは改行文字で終わります。
S370V	可変長 S370 レコード形式(V)。
S370VB	可変長ブロック S370 レコード形式(VB)。
S370VBS	スパンレコード可変長ブロック S370 レコード形式(VBS)。
デフォルト	グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。最大長は 1G です。

### SCANOVER

INPUT ステートメントが、@'character-string'式に指定された文字列を検出するまで、入力データレコードをスキャンするようにします。

**操作** MISCOVER、TRUNCOVER、および STOPOVER オプションを使用すると、INPUT ステートメントが@'character-string'式をスキャンしてレコードの終端に到達した場合に、その INPUT ステートメントがどのように動作するかを変更できます。デフォルト(FLOWOVER オプション)では、INPUT ステートメントは次のレコードをスキャンします。それ以外のオプションでは、INPUT ステートメントはスキャンを中止します。

**ヒント** SCANOVER と FLOWOVER の両方を指定することは冗長です。

**参照項目:** “[行の終わりを越えて読み込む](#)” (165 ページ)、[FLOWOVER](#) (153 ページ)、[MISCOVER](#) (155 ページ)、[STOPOVER](#) (158 ページ)、および [TRUNCOVER](#) (158 ページ)

**例** “[例 3: 指定の文字列の直後に記述された可変長レコードをスキャンする](#)” (170 ページ)

### SHAREBUFFERS

FILE ステートメントと INFILE ステートメントが同じバッファを共有することを指定します。

**別名** SHAREBUFS

**ヒント** INFILE、FILE、PUT ステートメントで SHAREBUFFERS オプションを使用すると、外部ファイルを直接更新できます。外部ファイルを直接更新すると、PUT ステートメント出力が出力バッファを経由せずに入力バッファから外部ファイルへと直接書き込まれるため、CPU 時間を節約できます。

また、SHAREBUFFERS を使用すると、レコード全体ではなく、外部ファイル内の特定のフィールドを更新できます。

**例** “[例 6: 外部ファイルの更新](#)” (172 ページ)

**注意** **SHAREBUFFERS、RECFM=V、\_INFILE\_**を使用する際、同一の長さのレコードを読み込んだ後、異なる長さのレコードを含むファイルを更新する場合には注意が必要です。レコードの長さを変更するには、**\_INFILE\_**を使用します。この潜在的な問題を回避する方法の1つとして、元々のレコード長が維持されるように**\_INFILE\_**を追加または削除することが考えられます。

### **START=variable**

変数名を指定します。この変数の値は、**PUT \_INFILE\_**ステートメントが書き込むレコードの最初の列番号として使用されます。自動変数と同じように、**START=**に指定した変数はデータセットに書き込まれません。

参照項目: [PUT ステートメントの\\_INFILE\\_オプション \(320 ページ\)](#)

### **STOPOVER**

**INPUT** ステートメントで、ステートメント内のすべての変数に対する値が見つからずに現在のレコードの終わりに達した場合、**DATA** ステップの処理が停止します。期待された数の値が入力行に含まれていない場合、**STOP** ステートメントが実行されないならば、**SAS** は **\_ERROR\_** に値 1 を設定してデータセットの作成を中止し、不完全なデータ行を出力します。

**ヒント** デフォルトの動作に戻すには **FLOWOVER** を使用します。

**参照項目:** [“行の終わりを越えて読み込む” \(165 ページ\)](#)、[FLOWOVER \(153 ページ\)](#)、[MISSOVER \(155 ページ\)](#)、[SCANOVER \(157 ページ\)](#)、および [TRUNCOVER \(158 ページ\)](#)

**例** [“例 2: 欠損値と短いレコードをリスト入力処理する” \(169 ページ\)](#)

### **TERMSTR=**

**Linux** でフォーマットされたファイルでの行末を制御します。デフォルトでは、ラインフィードのみ、またはキャリッジリターンとラインフィードが行の終わりを示します。行末の区切り文字を明示的に指定するには、次のいずれかの値を指定します。

**CR** キャリッジリターン。

**CRLF** キャリッジリターン、ラインフィード。

**LF** ラインフィード。このパラメータは、フォーマットされたファイルの読み込みに使用されます。

### **TRUNCOVER**

入力データレコードの長さが **INPUT** ステートメントの期待する長さよりも短い場合の **INPUT** ステートメントのデフォルトの動作を無効化します。デフォルトでは、**INPUT** ステートメントは次の入力データレコードを自動的に読み込みます。 **TRUNCOVER** を指定すると、**INPUT** ステートメントの期待する長さよりも短いレコードが存在する場合、可変長のレコードを読み込むことができます。値を割り当てられていない変数には、欠損値が設定されます。

**ヒント** **TRUNCOVER** を使用すると、フィールドの長さが期待された長さよりも短い場合、入力バッファの内容を変数に割り当てることができます。

**参照項目:** [“行の終わりを越えて読み込む” \(165 ページ\)](#)、[FLOWOVER \(153 ページ\)](#)、[MISSOVER \(155 ページ\)](#)、[SCANOVER \(157 ページ\)](#)、および [STOPOVER \(158 ページ\)](#)

例 “例 3: 指定の文字列の直後に記述された可変長レコードをスキャンする” (170 ページ)

## UNBUFFERED

バッファリングされた(先行)読み込みを実施しないよう指示します。

別名 UNBUF

操作 UNBUFFERED を指定すると、SAS は END=変数に値 1 を設定しなくなります。

ヒント DATALINES ステートメントを使用して入力ストリームデータを読み込む場合、UNBUFFERED が有効となります。

## INFILE=variable

この INFILE ステートメントの現在の入力バッファの内容を参照する文字変数を指定します。この変数は他の変数と同じように使用できます。また、値を割り当てることもできます。この変数の値は自動的に保持されます。初期値はブランクです。自動変数と同じように、INFILE=に指定した変数はデータセットに書き込まれません。

制限事項 *variable* には、すでに定義済みの変数は指定できません。INFILE=オプションには、その DATA ステップ内で初めて使用する変数を指定する必要があります。LENGTH ステートメントや ATTRIB ステートメントを使用して、INFILE=に指定した変数の長さを設定したり変更することはできません。ただし、ATTRIB ステートメントや FORMAT ステートメントを使用して、この変数に出力形式を指定することができます。

操作 この文字変数の最大長は、指定した INFILE ステートメントの論理レコード長(LRECL=(154 ページ))になります。ただし、プログラムの実行直前まで、SAS がこのファイルを開いて LRECL=オプションの値を確認することはありません。そのため、コンパイル実行中は、この変数のサイズは 32,767 バイトになります。

ヒント この変数の内容に対する変更は、INFILE ステートメントの現在の入力バッファの内容にただちに反映されます。バッファの変更後に、その INFILE ステートメントが有効なスコープで PUT INFILE ステートメントを実行すると、変更されたバッファの内容が反映されます。N=オプションで複数のバッファを指定した場合でも、INFILE=に指定した変数は、指定した INFILE ステートメントの現在の入力バッファにのみアクセスできます。

別のステートメントで INFILE=オプションを使用せずに入力バッファの内容にアクセスするには、自動変数 INFILE を使用します。

INFILE 変数は固定長ではありません。INFILE 変数に値を割り当てると、その変数の長さは、割り当てられた値の長さへと変更されます。

参照項目: “入力バッファのコンテンツへのアクセス” (160 ページ)

例 “例 9: 入力バッファ内のレコードを操作する” (173 ページ)

“例 10: 複数のファイルの入力バッファへのアクセス” (174 ページ)

## 詳細

### INFILE ステートメントの使用

INFILE ステートメントは読み込み対象となるファイルを指定するものであるため、入力データレコードを読み込む INPUT ステートメントよりも前に実行する必要があります。INFILE ステートメントは実行ステートメントであるため、IF-THEN ステートメントのような条件付き処理で INFILE ステートメントを使用できます。INFILE ステートメントを使用すると、入力データレコードのソースを制御できます。

通常、INFILE ステートメントは、外部ファイルからデータを読み込む場合に使用します。データをジョブストリームから読み込む場合、DATALINES ステートメントを使用する必要があります。ただし、INFILE ステートメントでのみ使用可能なデータ読み込みオプションを使用する場合、同じ DATA ステップ内で、DATALINES および DATALINES ステートメントのファイル指定を含む INFILE ステートメントを使用できません。詳細については、“[長い入力ストリームデータレコードの読み込み](#)” (165 ページ)を参照してください。

複数の INFILE ステートメントを使用して同じファイル指定を行い、各 INFILE ステートメントでオプションを使用した場合、それらのオプションが追加されません。混乱を避けるために、特定の外部ファイルに関するオプションはすべて最初の INFILE ステートメントで指定するようにしてください。

### 複数の入力ファイルの読み込み

DATA ステップの 1 回の反復で複数の入力ファイルから読み込みを行うには、次の 2 つの方法のうちどちらかを使用します。

- 複数のファイルを開いた状態に保ち、読込先のファイルを変更し、複数の INFILE ステートメントを使用する
- 単一の DATA ステップ内で現在の入力ファイルを動的に変更し、INFILE ステートメントで FILEVAR=オプションを使用する FILEVAR=オプションを使用すると、1 つのファイルから読み込みを行い、そのファイルを閉じた後、別のファイルを開くことができます。“[例 5: 複数の入力ファイルからの読み込み](#)” (171 ページ)を参照してください。

### 外部ファイルの直接更新

INFILE ステートメントと FILE ステートメントを組み合わせることで、外部ファイルに含まれているレコードを更新できます。これを行うには次のようになります。

1. FILE ステートメントの前に INFILE ステートメントを指定します。
2. 各ステートメントで、同一のファイル参照名または物理ファイル名を指定します。
3. FILE ステートメントのかわりに、INFILE ステートメントに、FILE ステートメントと INFILE ステートメントの両方で使用できるオプションを使用します。(これらのオプションを FILE ステートメントに指定しても無視されます。)

“[例 6: 外部ファイルの更新](#)” (172 ページ)を参照してください。

レコード全体ではなく、レコード内の個々のフィールドを更新する方法については、[SHAREBUFFERS オプション](#) (157 ページ)を参照してください。

### 入力バッファのコンテンツへのアクセス

`_INFILE_`変数を使用する以外に、`_INFILE_`自動変数を使用することでも、最後に実行した INFILE ステートメントの入力バッファの内容を参照できます。この

文字変数の値は自動的に保持されます。初期値は空白です。他の自動変数と同じように、\_INFILE\_はデータセットに書き込まれません。

INFILE ステートメントで \_INFILE\_=オプションを指定すると、この変数の値を自動変数 \_INFILE\_ でも間接的に参照できます。自動変数 \_INFILE\_ が存在する場合に、特定の INFILE ステートメントで \_INFILE\_ を省略すると、その INFILE ステートメント用の内部的な \_INFILE\_ 変数が作成されます。これ以外の場合に、特定の FILE に対して \_INFILE\_ 変数が作成されることはありません。

実行時または参照時に、この文字変数の最大長は、現在の \_INFILE\_ 変数の最大長と一致します。ただし、\_INFILE\_ が参照している変数の長さは実行時まで確定されないため、\_INFILE\_ の長さはコンパイル時に 32,767 バイトに設定されます。たとえば、\_INFILE\_ を長さが定義されていない新しい変数に割り当てる場合、新しい変数のデフォルトの長さは 32,767 バイトになります。LENGTH ステートメントや ATTRIB ステートメントを使用して、\_INFILE\_ の長さを設定または無効化することはできません。ただし、FORMAT ステートメントや ATTRIB ステートメントを使用すると、\_INFILE\_ に出力形式を割り当てることができます。

他の SAS 変数と同じように、割り当てステートメントを使用して、\_INFILE\_ 変数を更新することができます。また、PUT ステートメントの \_INFILE\_ でも出力形式を使用できます。たとえば、次の PUT ステートメントでは、16 進数の出力形式を使用して入力バッファの内容を出力します。

```
put _infile_ $hex100.;
```

\_INFILE\_ に変更を加えると、現在の INFILE ステートメントの入力バッファにただちに変更が反映されます。バッファの内容を変更した後、PUT \_INFILE\_ ステートメントを実行すると、変更したバッファの内容が出力されます。

N=オプションに複数のバッファを指定した場合でも、\_INFILE\_ は、その INFILE ステートメントで現在使用中の入力バッファの内容にのみアクセスします。N=に指定したすべてのバッファにアクセスできますが、特定のバッファを現在の入力バッファに設定するには、INPUT ステートメントに#行ポインタコントロールを指定する必要があります。

### 区切り文字で区切られたデータ値の読み込み

デフォルトでは、リスト入力での入力データレコードの読み込みに使用される区切り文字は、1つの空白になります。DSD (delimiter-sensitive data) オプション、DELIMITER=オプション、DLMSTR=オプション、DLMSOPT=オプションは、リスト入力での区切り文字の取り扱い方法に影響を与えます。DELIMITER=または DLMSTR=オプションは、INPUT ステートメントで、リスト入力により読み取られるデータ値の区切り文字として、空白以外の文字を使用することを指定します。DSD オプションを有効にすると、INPUT ステートメントではデフォルトの区切り文字としてカンマを使用します。

また、DSD オプションを指定すると、2つの連続する区切り文字が欠損値として読み込まれるようになります。デフォルトでは、INPUT ステートメントは、連続する区切り文字を1つの単位として扱います。DSD オプションを指定すると、INPUT ステートメントは、連続する区切り文字をそれぞれ独立したものとして扱うようになります。このため、2つの連続する区切り文字で囲まれた空の文字列は欠損値として読み込まれます。区切り文字をカンマから別の値へと変更する場合、DELIMITER=または DLMSTR=オプションを使用します。

たとえば、次の DATA ステッププログラムでは、リスト入力を使用して、カンマで区切られたデータを読み込みます。2番目のデータ行には欠損値が含まれています。SAS では、リスト入力での連続する区切り文字の使用を許可しているため、この INPUT ステートメントでは欠損値を検出できません。

```
data scores;
  infile datalines delimiter=';';
```

```

input test1 test2 test3;
datalines;
91,87,95
97,,92
,1,1
;

```

FLOWOVER オプションが有効である場合、データセット SCORES には、3 つではなく 2 つのオブザベーションが含まれます。2 番目のオブザベーションは正しく作成されません。

OBS	TEST1	TEST2	TEST3
1	91	87	95
2	97	92	1

この問題を修正するには、INFILE ステートメントで DSD オプションを使用する必要があります。

```

data scores;
input test1 test2 test3;
datalines;
91,87,95
97,,92
,1,1
;

infile datalines dsd;

```

今回の INPUT ステートメントでは、2 つの連続する区切り文字を検出するため、2 番目のオブザベーションに含まれている変数 TEST2 には欠損値が割り当てられます。

OBS	TEST1	TEST2	TEST3
1	91	87	95
2	97	.	92
3	.	1	1

また、DSD オプションを指定すると、リスト入力で、引用符で囲まれた文字列内に区切り文字を含む文字値を読み込むことができるようになります。たとえば、データがカンマで区切られている場合、DSD オプションを指定すると、そのような文字列を引用符で囲むことで、同文字列に含まれているカンマを有効な文字として読み込むことができます。この場合の引用符は、文字値の一部としては保存されません。引用符を文字値の一部として保持するには、INPUT ステートメントでチルダ(~)フォーマット修飾子を使用します。[“例 1: 区切り文字の扱い方の変更” \(167 ページ\)](#)を参照してください。

**注:** テキストファイルがローカルのエンコーディング環境以外の場所で作成された場合、EBCDIC 環境または ASCII 環境のどちらかで ENCODING=オプションを指定する必要があります。たとえば、ASCII プラットフォーム上で EBCDIC ファイルを読み込む場合、INFILE ステートメントに ENCODING=オプションを指定することをお勧めします。ただし、INFILE ステートメントで DSD オプションおよび DLM オプションを使用する場合は、ENCODING=オ

プションを指定する必要があります。これは、2つのオプションがセッションエンコーディングの特定の文字(引用符、カンマ、空白など)を必要とするためです。エンコーディング固有の入力形式は、対応するバイナリファイルと組み合わせて使用する必要があります。このようなバイナリファイルには、文字フィールドと文字以外のフィールドが含まれています。

デフォルト長よりも大きい長さのデータを扱う場合、入力形式を指定する必要があります。たとえば、日付または時間値、名前、住所などは、8文字よりも長い場合があります。そのような場合、INFORMAT ステートメントを DATA ステップに追加するか、または入力形式を INPUT ステートメントに直接追加します。INPUT ステートメントに入力形式を追加する場合、その入力形式の前にコロン(:)を追加する必要があります。これにより、現在の区切り文字の後の最初の文字から、この入力形式内に指定された文字数までが読み込まれるようになります。

```
data a;
  infile 'c:\sas\example.csv' dlm='09'x dsd trunccover;
  input fname :$20. lname :$30. address1 :$50. address2 :$50. city :$40.
        state :$2. zip phone :$12.;
run;
```

区切り文字で区切られたデータを読み込む場合に考慮すべきその他の INFILE オプションを次に示します。

#### FIRSTOBS=

最初に読み込むオブザベーションの番号を指定します。たとえば、ヘッダーレコードをスキップする場合には、FIRSTOBS=2 と指定します。

#### TERMSTR=

行末文字を指定する場合に使用します。このオプションは、あるオペレーティングシステムで作成されたデータを、別のオペレーティングシステムと共有する場合に使用すると便利です。たとえば、Linux 環境で作業している場合に、Windows 環境で作成されたファイルを読み込む必要があるならば、TERMSTR=CRLF と指定します。

#### TRUNCCOVER=

現在のレコード内にある使用可能な入力データのみを使用して、できるだけ多くの変数を割り当てるよう指定します。デフォルトでは、INPUT ステートメントが、リストされている変数に割り当てる十分な数のデータを見つけられずにレコードの終端を超えて読み込む場合、同ステートメントは次のレコードからデータを継続して読み込みます。次のレコードを読み込まないようにするには、TRUNCCOVER=オプションを使用します。

トラブルシューティングの例をいくつか次に示します。

問題	解決策
<p>SAS ログに次のようなメッセージが出力されており、データセット内のすべてのデータが読み込み用に表示されています。 One or more lines were truncated.</p>	<p>多くの場合、これは問題にはなりません。このメッセージは、次の条件のいずれかが真である場合に出力されます。</p> <ol style="list-style-type: none"> <li>1. 最大レコード長がアクティブな論理レコード長より短い。</li> <li>2. すべてのデータがデータセット内に存在する。</li> <li>3. 1 よりも大きい FIRSTOBS=の値を使用している。</li> </ol> <p>このメッセージは、行の長さが LRECL=の値よりも大きいために、FIRSTOBS=オプションによりその行がスキップされたことを意味します。SAS は、行の長さが LRECL=の値よりも大きいことを認識しますが、FIRSTOBS=オプションによりその行がスキップされたことは認識しません。</p>
<p>最後の変数が数値変数であり、すべてのオブザベーションで、この変数が欠損値となっています。</p> <p>変数が欠損値となるよくある原因としては、改行文字としてラインフィード(LF)のみを使用する Linux 環境で Windows ファイルを読み込んだことが挙げられます。Linux オペレーティングシステムは、キャリッジリターン(CR)をデータとして読み込みます。しかし、CR は数値データではないため、無効なデータであるとのメッセージが生成されます。</p>	<p>INFILE ステートメントで TERMSTR=CRLF オプションを指定します。</p> <p>SAS 9 よりも前のリリースを使用している場合、ファイルを適切な Linux 構造に変換するか、またはキャリッジリターン(&lt;CR&gt;)を区切り文字のリストに追加します。</p> <p>CSV ファイルを読み込む場合、キャリッジリターンを DL DLM='2COD'x として指定します。</p> <p>タブで区切られたファイルを読み込む場合、DLM='090D'x を使用します。</p> <p>これら以外の区切り文字を使用している場合、SAS テクニカルサポートに連絡してください。</p>



## 問題

CSV ファイルを読み込むと、引用符で囲まれた文字列の内部でレコードが分割されます。たとえば、長いコメントフィールドが、読み込み中のファイル内の行の途中でいきなり中断され、残りの部分は次の行に続いている箇所があります。ファイルをテキストエディタで開くと、このことを確認できます。

この問題は、Microsoft Excel ワークシートからファイルを作成した場合によく発生します。Excel では、フォーマットを支援するために、列にソフトリターンが含まれています。ワークシートを CSV ファイルとして保存すると、SAS は、このソフトリターンを誤ってレコード終端マークとして解釈します。この結果、レコードが不正に分割されることとなります。

## 解決策

SAS 9 よりも前のリリースを使用している場合、次のプログラムを実行します。このプログラムは、二重引用符で囲まれているラインフィールド文字を空白へと変換し、SAS がファイルを正しく読み込めるようにします。このプログラムでは、INFILE ステートメントと FILE ステートメントで同じファイルを参照しています。

SHAREBUFFERS オプションを指定することにより、外部ファイルを直接更新して、問題のある文字を削除できるようにしています。

```
data _null;
  infile 'c:\_today\mike.csv' recfm=n sharebuffers;
  file 'c:\_today\mike.csv' recfm=n;
  input a $char1.;
  retain open 0;
/* toggle the open flag */
  if a="" then open=not open;
  if a='0A'x and open then put ' ';
run;
```

このプログラムは、ファイルを 1 バイトずつ読み込み、必要ならばラインフィールド文字を置き換えます。

### 長い入力ストリームデータレコードの読み込み

INFILE ステートメントと DATALINES ファイル指定を組み合わせることで、入力ストリームデータを処理できます。INPUT ステートメントは、DATALINES ステートメントの後に記述されたデータレコードを読み込みます。CARDIMAGE システムオプションを指定した場合、またはこのオプションがお使いのシステムでデフォルトとなっている場合、ブランクが埋め込まれた 80 バイトのパンチカードイメージのようにデータ行が処理されます。INFILE ステートメントではデフォルトで FLOWOVER オプションが使用されるため、INPUT ステートメントは、同ステートメントに含まれているすべての変数用の値を現在のレコード内で検出できない場合、次のレコードを読み込みます。データが正しく処理されることを保証するには、レコード長が 80 バイトを超える場合の入力では外部ファイルを使用します。

注: NOCARDIMAGE システムオプション([“CARDIMAGE System Option” \(SAS Viya System Options: Reference\)](#)を参照)は、データ行を 80 バイトのカードイメージとして扱わないよう指定します。データ行の終端は、引用符で囲まれた文字列の場合を除いて、常に最後のトークンの終端として扱われます。

### 行の終わりを超えて読み込む

デフォルトでは、INPUT ステートメントが現在の入力データレコードの終端を超えて読み込もうとする場合、同ステートメントは入力ポインタを次のレコードの列 1 に移動してから、残りの値を読み込みます。このデフォルトの動作は、FLOWOVER オプションにより指定されます。この結果、次のメッセージが SAS ログに書き込まれます。

NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

行の終端に到達した場合の INPUT ステートメントの動作を変更する方法として、複数のオプションを使用できます。STOPOVER オプションを指定すると、この状態をエラーと判断し、データセットの作成を終了します。MISSOVER および TRUNCOVER オプションを指定すると、現在の INPUT ステートメントが満たされない場合でも、入力ポインタが次の行へと移動しなくなります。SCANOVER オプションを '@character-string' とともに使用すると、指定した *character-string* が検出されるまで入力レコードがスキャンされます。デフォルトの動作に戻すには FLOWOVER オプションを使用します。

TRUNCOVER と MISSOVER は類似したオプションです。MISSOVER オプションを指定すると、INPUT ステートメントは、同ステートメント内で指定されたフィールド長よりも値の長さが小さいためにフィールド全体を読み込めない場合、その値を欠損値に設定します。TRUNCOVER オプションを指定すると、対応する変数に読み込まれた文字が書き込まれます。

たとえば、可変長レコードを含む外部ファイルに次のレコードが含まれているとします。

```
----+----1----+----2
1
22
333
4444
55555
```

次の DATA ステップは、このデータを読み込み、SAS データセットを作成します。これらの入力レコードのうち、変数 TESTNUM の長さ(入力形式でフォーマットされたもの)と同じ長さのレコードは 1 つだけです。

```
data numbers;
  infile 'external-file';
  input testnum 5.;
run;
```

デフォルトでは入力レコードの読み込み時に FLOWOVER オプションが使用されるため、この DATA ステップは、5 つの入力レコードから 3 つのオブザベーションを作成します。

INFILE ステートメントで MISSOVER オプションを指定すると、この DATA ステップは 5 つのオブザベーションを作成します。短すぎるレコードから読み込まれた値はすべて欠損値に設定されます。短すぎるために現在の INPUT ステートメントを満たすことができないレコード内にどんな値が含まれているかを確認したい場合は、INFILE ステートメントで TRUNCOVER オプションを指定します。

```
infile 'external-file' truncover;
```

この場合、DATA ステップは、同じ入力レコードを読み込み、5 つのオブザベーションを作成します。作成された SAS データセットを比較した表を次に示します。

表 2.4 INFILE ステートメントで異なるオプションを使用した場合の TESTNUM の値

OBS	FLOWOVER	MISSOVER	TRUNCOVER
1	22	.	1
2	4444	.	22
3	55555	.	333

OBS	FLOWOVER	MISSOVER	TRUNCOVER
4		.	4444
5		55555	55555

## 比較

- INFILE ステートメントでは、DATA ステップの INPUT ステートメントに対して入力ファイルが指定されます。FILE ステートメントでは、DATA ステップの PUT ステートメントに対して出力ファイルが指定されます。
- 通常、INFILE ステートメントは、外部ファイルからデータを読み込む場合に使用します。DATALINES ステートメントは、ジョブストリーム内の同ステートメントの後にデータが続くことを指定します。INFILE ステートメントと DATALINES ファイル指定を組み合わせると、INPUT ステートメントが入カストリームデータを読み込む方法に影響を与える特定のデータ読み込みオプションを利用できるようになります。

## 例

### 例 1: 区切り文字の扱い方の変更

デフォルトでは、INPUT ステートメントは空白を区切り文字として使用します。次のデータステップでは、カンマを区切り文字として使用します。

```
data num;
  infile datalines dsd;
  input x y z;
  datalines;
,2,3
4,5,6
7,8,9
;
```

INFILE ステートメントの DATALINES 引数を指定すると、入力ストリームデータ行を読み込むための INFILE ステートメントオプションを使用できるようになります。DSD オプションを指定すると、カンマがデフォルトの区切り文字に設定されます。最初のデータ行の先頭にある値の前にカンマが記述されているため、1 番目のオブザベーション内の変数 X には欠損値が割り当てられ、変数 Y には値 2 が割り当てられます。

データで複数の区切り文字を使用している場合や、カンマ以外の単一区切り文字を使用している場合、DELIMITER=オプションを使用してその区切り文字値を指定します。次の例では、文字 **a** および **b** が区切り文字として使用されています。

```
data nums;
  infile datalines dsd delimiter='ab';
  input X Y Z;
  datalines;
1aa2ab3
4b5bab6
7a8b9
;

proc print; run;
```

実行後に作成された NUM データセットを PROC PRINT で出力した結果を次に示します。DSD オプションを指定すると、リスト入力は2つの連続する区切り文字を検出できるようになります。このため、1番目と2番目のオブザベーション内の変数には欠損値が割り当てられます。DSD オプションを省略すると、文字 a、b、aa、ab、ba、bb が区切り文字として使用されるため、変数には欠損値が割り当てられません。

#### アウトプット 2.6 NUM データセット

Obs	X	Y	Z
1	1	.	2
2	4	5	.
3	7	8	9

文字列を区切り文字として使用するには、DLMSTR=オプションで区切り文字値を指定します。次の例では、文字列 **PRD** を区切り文字として使用します。この文字列にはすべて大文字が含まれていることに注意してください。DLMSOPT=オプションを使用することで、**PRD**、**Prd**、**PRd**、**PrD**、**pRd**、**pRD**、**prD**、**prd** をすべて有効な区切り文字にしています。

```
data test;
  infile datalines dsd dlmstr='PRD' dlmsopt='i';
  input X Y Z;
  datalines;
1PRD2PRd3
4PrD5Prd6
7pRd8pRD9
;
```

PROC PRINT による出力は、TEST データセット内のすべてのオブザベーションを表示します。

#### アウトプット 2.7 TEST データセット

Obs	X	Y	Z
1	1	2	3
2	4	5	6
3	7	8	9

次の DATA ステップでは、修飾リスト入力と DSD オプションを使用して、カンマで区切られたデータや、文字値の一部としてカンマを含んでいる可能性のあるデータを読み込みます。

```
data scores;
  infile datalines dsd;
  input Name : $9. Score
        Team : $25. Div $;
```

```

datalines;
Joseph,76,"Red Racers, Washington",AAA
Mitchel,82,"Blue Bunnies, Richmond",AAA
Sue Ellen,74,"Green Gazelles, Atlanta",AA
;

```

```
proc print; run;
```

実行後に作成された SCORES データセットを PROC PRINT で出力した結果を次に示します。区切り文字(カンマ)は、変数 TEAM の値の一部として格納されます。一方、引用符はそのような値としては格納されません。

#### アウトプット 2.8 SCORES データセット

Obs	Name	Score	Team	Div
1	Joseph	76	Red Racers, Washington	AAA
2	Mitchel	82	Blue Bunnies, Richmond	AAA
3	Sue Ellen	74	Green Gazelles, Atlanta	AA

#### 例 2: 欠損値と短いレコードをリスト入力で処理する

次の例では、リスト入力を使用してデータを読み込む場合に、欠損値を原因とする問題の発生を防ぐ方法を示します。この例の最初のデータ行には、5つの値ではなく、3つの値だけが含まれています。MISSOVER オプションを指定すると、最初のデータ行の4番目と5番目の値は欠損値に設定されます。

```

data weather;
  infile datalines missover;
  input temp1-temp5;
  datalines;
97.9 98.1 98.3
98.6 99.2 99.1 98.5 97.5
96.2 97.3 98.3 97.6 96.5
;

```

SAS は、最初のデータ行に含まれている3つの値を、TEMP1、TEMP2、TEMP3の値として読み込みます。MISSOVER オプションを指定すると、最初のオブザベーション内の変数 TEMP4 および TEMP5 の値が欠損値に設定されます。これらの変数用の値が、現在の入力データレコード内には提供されていないためです。

MISSOVER オプションを省略するか、または FLOWOVER オプションを指定すると、SAS は入力ポインタを2番目のデータ行に移動し、変数 TEMP4 および TEMP5 の値をその行から読み込みます。DATA ステップの次の実行では、新しい行として、3番目の行が読み込まれます。SAS ログには次のメッセージが表示されます。

```
NOTE: SAS went to a new line when INPUT statement
reached past the end of a line.
```

INFILE ステートメントに STOPOVER オプションを使用することもできます。STOPOVER オプションを使用すると、INPUT ステートメントが生データのレコード内に十分な値を検出できない場合には、DATA ステップが実行を停止するようになります。

```
infile datalines stopover;
```

最初のデータレコードでは TEMP4 の値が見つからないため、SAS は `_ERROR_` に値 1 を設定した後、データセットの作成を中止し、データ行 1 を出力します。

### 例 3: 指定の文字列の直後に記述された可変長レコードをスキャンする

次の例では、TRUNCOVER オプションと SCANOVER オプションを組み合わせることで、電話帳から電話番号を取り出します。電話番号の前には、必ず "phone:" という文字列が記述されているものとします。電話番号は国際番号形式であるため、最大長は 32 文字になります。

```
filename phonebk host-specific-path;
data _null_;
  file phonebk;
  input line $80.;
  put line;
  datalines;
  Jenny's Phone Book
  Jim Johanson phone: 619-555-9340
    Jim wants a scarf for the holidays.
  Jane Jovalley phone: (213) 555-4820
    Jane started growing cabbage in her garden.
    Her dog's name is Juniper.
  J.R. Hauptman phone: (49)12 34-56 78-90
  J.R. is my brother.
  ;
run;
```

@'phone:' を使用すると、電話番号を含むファイルの行をスキャンし、電話番号が始まる位置にファイルポインタを移動できます。また、TRUNCOVER オプションと SCANOVER オプションを組み合わせることで、文字列 'phone:' を含んでいない行をスキップし、電話番号のみをログに書き込みます。

```
data _null_;
  infile phonebk truncover scanover;
  input @'phone:' phone $32.;
  put phone=;
run;
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
phone=619-555-9340 phone=(213) 555-4820 phone=(49)12 34-56 78-90
```

### 例 4: 可変長レコードを含むファイルの読み込み

次の例では、LENGTH= オプションと \$VARYING. 入力形式を組み合わせることで、可変長レコードを含むファイルを読み込んでいます。

```
data a;
  infile file-specification length=linelen lrecl=510 pad;
  input firstvar 1-10 @; /* assign LINELEN */
  varlen=linelen-10; /* Calculate VARLEN */
  input @11 secondvar $varying500. varlen;
run;
```

この DATA ステップでは次のことが行われます。

- INFILE ステートメントでは、変数 LINELEN を作成しますが、この変数には値を割り当てません。

- 最初の INPUT ステートメントを実行すると、SAS はレコード行の長さを判定し、変数 LINELEN に値を割り当てます。後置@が指定されているため、次の INPUT ステートメントの入力バッファにレコードが保持されます。
- 割り当てステートメントでは、2つの既知の長さ(FIRSTVAR の長さとレコード全体の長さ)を使用して、VARLEN の長さを決定します。
- 2番目の INPUT ステートメントでは、\$VARYING500.入力形式付きの VARLEN 値を使用して、変数 SECONDVAR の値を読み込みます。

詳細については、“[\\$VARYING Informat](#)” (*SAS Viya Formats and Informats: Reference*)を参照してください。

### 例 5: 複数の入力ファイルからの読み込み

次の DATA ステップでは、DATA ステップの反復ごとに、2つの入力ファイルから読み込みます。SAS が1つのファイルから別のファイルへと処理を切り替える際に、各ファイルは開かれたままになります。入力ポインタは、現在の位置のまま移動しないため、次回 INPUT ステートメントを実行すると、ファイルの読み込みがその位置から開始されます。

```
data qtrtot(drop=jansale febsale marsale aprsale maysale junsale);
  /* identify location of 1st file */
  infile file-specification-1;
  /* read values from 1st file */
  input name $ jansale febsale marsale;
  qtr1tot=sum(jansale,febsale,marsale);
  /* identify location of 2nd file */
  infile file-specification-2;
  /* read values from 2nd file */
  input @7 aprsale maysale junsale;
  qtr2tot=sum(aprsale,maysale,junsale);
run;
```

SAS が最小の入力ファイルの終端に到達すると、DATA ステップは終了します。

次の DATA ステップでは、FILEVAR=オプションを使用して、DATA ステップの反復ごとにそれぞれ別のファイルを読み込みます。

```
data allsales;
  length fileloc myinfile $ 300;
  input fileloc $ ; /* read instream data */
  /* The INFILE statement closes the current file
  and opens a new one if FILELOC changes value
  when INFILE executes */
  infile file-specification filevar=fileloc
  filename=myinfile end=done;
  /* DONE set to 1 when last input record read */
  do while(not done);
  /* Read all input records from the currently */
  /* opened input file, write to ALLSALES */
  input name $ jansale febsale marsale;
  output;
end;
put 'Finished reading ' myinfile=;
datalines;
external-file-1
external-file-2
external-file-3
;
```

FILENAME=オプションは、現在の入力ファイルの名前を変数 MYINFILE に割り当てます。LENGTH ステートメントは、FILENAME=変数と FILEVAR=変数の長さ、このファイル名を値として格納するために十分であることを保証します。PUT ステートメントは、現在開かれている入力ファイルの物理名を SAS ログに出力します。

### 例 6: 外部ファイルの更新

次の例では、SHAREBUFFERS オプションを指定した INFILE ステートメントと、INPUT、FILE、PUT ステートメントを組み合わせることで、外部ファイルを直接更新します。

```
data _null_;
  /* The INFILE and FILE statements */
  /* must specify the same file. */
  infile file-specification-1 sharebuffers;
  file file-specification-1;
  input state $ 1-2 phone $ 5-16;
  /* Replace area code for NC exchanges */
  if state= 'NC' and substr(phone,5,3)='333' then
    phone='910-'||substr(phone,5,8);
  put phone 5-16;
run;
```

### 例 7: コピーされたレコードの切り捨て

LENGTH=オプションは、PUT \_INFILE\_ ステートメントを使用して、入力ファイルを別のファイルにコピーする場合に便利です。LENGTH=オプションを指定すると、コピーされたレコードを切り捨てることができます。たとえば、次のステートメントでは、入力データレコードを出力ファイルに書き込む前に、各入力データレコードの末尾から 20 列を切り捨てます。

```
data _null_;
  infile file-specification-1 length=a;
  input;
  a=a-20;
  file file-specification-2;
  put _infile_;
run;
```

PUT \_INFILE\_ ステートメントでコピーした内容を切り捨てる場合、START=オプションを使用すると便利です。たとえば、各レコードの先頭から 10 列目までをコピーしたくない場合、次のステートメントを使用すると、入力バッファ内の各レコードの 11 列目から末尾までをコピーできます。

```
data _null_;
  infile file-specification start=s;
  input;
  s=11;
  file file-specification-2;
  put _infile_;
run;
```

### 例 8: ポインタ位置の出力

次の DATA ステップでは、入力バッファ内の現在のポインタの位置を表す値を、変数 LINEPT および COLUMNPT に割り当てます。

```
data _null_;
  infile datalines n=2 line=Linept col=Columnpt;
```



```

input name $ 1-15 #2 @3 id;
put linept= columnpt=;
datalines;
J. Brooks
40974
T. R. Ansen
4032
;

```

これらのステートメントは、DATA ステップを実行するたびに、次の行を出力します。これは、PUT ステートメントの実行時には、入力ポインタが入力バッファ内の 2 番目の行に存在するためです。

```

Linept=2 Columnpt=9
Linept=2 Columnpt=8

```

### 例 9: 入力バッファ内のレコードを操作する

`_INFILE` 変数には、常に、INPUT ステートメントを通じて読み込まれた最後のレコードが含まれています。次の例では、`_INFILE` 変数を使用して各種のタスクを実施する方法を示します。

- INPUT ステートメントを使用せずに解析したいレコード全体を読み込む方法を示します。
- SAS ログに書き込みたいレコード全体を読み込みます。
- この行が INPUT ステートメントにより解析される前に、入力レコードの内容を変更します。

次の例では、電話料金情報を含むファイルを作成します。数値データ(minutes および charge)は、山かっこ(<>)で囲んで指定します。

```

filename phonbill host-specific-filename;
data _null_;
  file phonbill;
  input line $80.;
  put line;
  datalines;
  City Number Minutes Charge
  Jackson 415-555-2384 <25> <2.45>
  Jefferson 813-555-2356 <15> <1.62>
  Joliet 913-555-3223 <65> <10.32>
  ;
run;

```

次のコードでは、各レコードを読み込み、レコードを解析することにより、minute と charge の値を取り出します。

```

data _null_;
  infile phonbill firstobs=2;
  input;
  city = scan(_infile_, 1, ' ');
  char_min = scan(_infile_, 3, ' ');
  char_min = substr(char_min, 2, length(char_min)-2);
  minutes = input(char_min, BEST12.);
  put city= minutes=;
run;

```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
city=jackson minutes=25 city=jefferson minutes=15 city=joliet minutes=65
```

次に示すコード内の INPUT ステートメントは、このファイルからレコードを読み込みます。\_INFILE\_自動変数を PUT ステートメントで使用するにより、レコードをログに書き込んでいます。

```
data _null_;
  infile phonbill;
  input;
  put _infile_;
run;
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
City Number Minutes Charge Jackson 415-555-2384 <25> <2.45> Jefferson 813-555-2356 <15> <1.62>
Joliet 913-555-3223 <65> <10.32>
```

次のコードでは、最初の INPUT ステートメントで、読み込んだレコードを入力バッファ内に保持します。\_INFILE\_=オプションにより、数値データから山かっこ(<>)を取り除きます。2 番目の INPUT ステートメントで、バッファ内の値を解析します。

```
data _null_;
  length city number $16. minutes charge 8;
  infile phonbill firstobs=2;
  input @;
  _infile_ = compress(_infile_, '<>');
  input city number minutes charge;
  put city= number= minutes= charge=;
run;
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
city=jackson number=415-555-2384 minutes=25 charge=2.45 city=jefferson number=813-555-2356
minutes=15 charge=1.62 city=joliet number=913-555-3223 minutes=65 charge=10.32
```

### 例 10: 複数のファイルの入力バッファへのアクセス

次の例では、\_INFILE\_自動変数と \_INFILE\_=オプションの両者を使用して複数のファイルを読み込んだ後、それぞれの入力バッファにアクセスしています。次のプログラムを実行すると 4 つのファイルが作成されます。そのうちの 3 つはデータファイルであり、4 つ目のファイルにはすべてのデータファイルの名前が含まれます。2 番目の DATA ステップでは、このファイル名ファイルを読み込み、各データファイルを開き、その内容をログに書き込みます。PUT ステートメントでは、ファイル名ファイルとデータファイル用に \_INFILE\_ 変数が必要となるため、\_INFILE\_ 変数の 1 つは *fname* を使用して参照されます。

```
data _null_;
  do i = 1 to 3;
    fname = 'external-data-file' || put(i,1.) || '.dat';
    file datfiles filevar=fname;
    do j = 1 to 5;
      put i j;
    end;
    file 'external-filenames-file';
    put fname;
  end;
run;
```

```

data _null_;
  infile 'external-filenames-file' _infile_=fname;
  input;
  infile datfiles filevar=fname end=eof;
  do while(^eof);
    input;
    put fname _infile_;
  end;
run;

```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```

NOTE: The infile 'external-filenames-file' is: File Name=external-filenames-file, RECFM=V, LRECL=256 NOTE:
The infile DATFILES is: File Name=external-data-file1.dat, RECFM=V, LRECL=256 external-data-file1.dat 1 1
external-data-file1.dat 1 2 external-data-file1.dat 1 3 external-data-file1.dat 1 4 external-data-file1.dat 1 5
NOTE: The infile DATFILES is: File Name=external-data-file2.dat, RECFM=V, LRECL=256 external-data-
file2.dat 2 1 external-data-file2.dat 2 2 external-data-file2.dat 2 3 external-data-file2.dat 2 4 external-data-
file2.dat 2 5 NOTE: The infile DATFILES is: File Name=external-data-file3.dat, RECFM=V, LRECL=256
external-data-file3.dat 3 1 external-data-file3.dat 3 2 external-data-file3.dat 3 3 external-data-file3.dat 3 4
external-data-file3.dat 3 5

```

### 例 11: 外部ファイル読み込み時のエンコードの指定

この例では、外部ファイルから SAS データセットを作成します。外部ファイルのエンコーディングは UTF-8 であり、現在の SAS セッションのエンコーディングは Wlatin1 です。デフォルトでは、SAS は、外部ファイルのエンコーディングがセッションエンコーディングと同じであると見なします。このため、文字データは新しい SAS データセットに正しく書き込まれません。

外部ファイルの読み込み時に使用するエンコーディングを指定するには、ENCODING=オプションを指定します。外部ファイルのエンコーディングを UTF-8 に指定すると、新しい SAS データセットへの書き込み時に、外部ファイルが UTF-8 から現在のセッションエンコーディングにトランスコードされます。これで、新しいデータセットにデータが Wlatin1 のエンコーディングで正しく書き込まれるようになります。

```

libname myfiles 'SAS-library';
filename extfile 'external-file';
data myfiles.unicode;
  infile extfile encoding='utf-8';
  input Make $ Model $ Year;
run;

```

### 関連項目:

#### ステートメント:

- [“FILENAME ステートメント” \(96 ページ\)](#)
- [“INPUT ステートメント” \(179 ページ\)](#)
- [“LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide](#)
- [“PUT ステートメント” \(318 ページ\)](#)

#### システムオプション:

- [“LOCKDOWN System Option in SAS Intelligence Platform: Application Server Administration Guide](#)

## INFORMAT ステートメント

変数に形式を関連付けます。

**該当要素:** DATA ステップまたは PROC ステップ

**カテゴリ:** CAS  
情報

**種類:** 宣言

### 構文

**INFORMAT** *variable-1* <...*variable-n*> <*informat*>;

**INFORMAT** <*variable-1*> <... *variable-n*> <DEFAULT=*default-informat*>;

**INFORMAT** *variable-1* <...*variable-n*> *informat* <DEFAULT=*default-informat*>;

### 引数

#### *variable*

1 つまたは複数の変数に関連付ける形式を指定します。*informat* を指定する場合、または他の引数を指定しない場合は、少なくとも *variable* を 1 つ指定する必要があります。形式として DEFAULT= を使用する場合、変数の指定は任意です。

**ヒ** 変数から形式の関連付けを取り消すには、この変数を INFORMAT  
**ン** ステートメントに形式を指定せずに使用します。INFORMAT ステ  
**ト** ートメントは SET ステートメントの後に指定します。“[例 3: 形式の取り消し](#)” (179 ページ) を参照してください。

#### *informat*

INFORMAT ステートメントにリストされている変数の値を読み込むときに使用する形式を指定します。

**ヒ** 形式が INFORMAT ステートメントを使用して変数に関連付けら  
**ト** れ、INPUT ステートメントにて同じ変数に同じ形式が関連付けられていない場合、この形式は INPUT ステートメントにコロン(:)修飾子付きで指定された形式と同じように動作します。変数は、形式を使いリスト経由で読み込まれます。たとえば、コロン修飾子を入力形式とともに使用すると、8 バイトよりも長い文字の値や非標準の値を含む数値を読み込むことができます。詳細については、“[INPUT ステートメント、リスト](#)” (204 ページ) を参照してください。

**参照項目:** [SAS Viya Formats and Informats: Reference](#)

**例** “[例 2: 数値および文字の形式を指定する](#)” (178 ページ)

#### **DEFAULT=** *default-informat*

INFORMAT ステートメントにリストされている変数の値を読み込むときに使用する一時的なデフォルトの形式を指定します。*variable* を指定しない場合、DATA ステップにあるすべての変数の読み込みに使用する一時的なデフォルトの形式として、DEFAULT= に指定した形式が適用されます。数値変数には数値の形式が適用され、文字変数には文字の形式

が適用されます。このデフォルト入力形式は、現在の DATA ステップにのみ適用されます。

DEFAULT=に指定した入力形式の適用対象

- INFORMAT または ATTRIB ステートメントに指定されていない変数
- SAS データセット内で入力形式が恒久的に関連付けられていない変数
- 現在の DATA ステップにおいて、明示的に指定した入力形式を使用して読み込まれない変数

**デフォルト** DEFAULT=の指定を省略すると、デフォルトの数値の入力形式として  $w.d$  が使用され、デフォルトの文字の入力形式として  $\$w.$  が使用されます。

**制限事項** この引数は DATA ステップのみで使用します。

**ヒント** DEFAULT=は、INFORMAT ステートメントのどの位置に指定してもかまいません。数値に対するデフォルト値、文字に対するデフォルト値、またはその両方を指定できます。

**例** [“例 1: デフォルトの入力形式を指定する” \(178 ページ\)](#)

## 詳細

### 基本

DATA ステップ内の INFORMAT ステートメントは、変数に入力形式を恒久的に割り当てます。標準の SAS 入力形式または PROC FORMAT で定義したユーザー定義の入力形式を指定できます。1 つの INFORMAT ステートメントにて、同じ入力形式を複数の変数に関連付けることや、異なる入力形式を複数の変数に関連付けることができます。同じ変数が複数の INFORMAT ステートメントに指定されている場合、最後に割り当てられた入力形式が使用されます。

#### 注意:

**INFORMAT ステートメントは事前に定義されていない文字変数の長さを定義するため、SET ステートメントの前に使用すると、DATA ステップ内の文字変数の値を切り捨てることができます。**

### INFORMAT ステートメントで入力形式を割り当てた場合の変数の処理

INFORMAT ステートメントを使用して変数に入力形式を割り当てると、修飾リスト入力で使用される入力形式と同じように動作します。変数はリスト入力の走査機能を使用して読み込まれますが、入力形式が適用されます。修飾リスト入力では、SAS は次の処理を行います。

- 入力形式の  $w$  の値を使用して、外部ファイルの列位置や入力フィールド幅を設定しない
- 入力形式の  $w$  の値を使用して、事前に定義されていない文字変数の長さを設定する
- 数値の入力形式の  $w$  の値は無視する
- 入力形式の  $d$  の値を、数値入力形式の場合と同じように使用する
- 入力データに埋め込まれている空白は、INFILE ステートメントの DLM= オプションまたは DLMSTR= オプションを変更しない限り、区切り文字として処理する

フォーマット入力やカラム入力などの別の入力スタイルを使用するように INPUT ステートメントをコード化した場合、INFORMAT オプションの使用時にはその入力スタイルは使用されません。

## 比較

- ATTRIB ステートメントと INFORMAT ステートメントは、どちらも変数に入力形式を関連付けることができます。また、どちらのステートメントを使用しても変数に関連付けられた入力形式を変更することができます。また、PROC DATASETS で INFORMAT ステートメントを使用することで、変数に関連付けられている入力形式の変更や削除が行えます。
- ステートメントを使用すると、指定した変数が含まれる SAS データセットのディスクリプタ情報が変更されます。一部の PROC ステップでも INFORMAT ステートメントを使用できますが、ルールは異なります。詳細については、“[FORMAT Procedure](#)” (*SAS Viya Data Management and Utility Procedures Guide*)を参照してください。

## 例

### 例 1: デフォルトの入力形式を指定する

次の例では、INFORMAT ステートメントを使用してデフォルトの数値入力形式を関連付けます。

```
data tstinfmt;
  informat default=3.1;
  input x;
  put x;
  datalines;
111
222
333
;
```

PUT ステートメントの実行結果は次のようになります。

```
11.1
22.2
33.3
```

### 例 2: 数値および文字の入力形式を指定する

この例では、SAS 変数に文字の入力形式と数値の入力形式を関連付けます。文字変数は列位置 15 まですべて占めるものではありませんが、INPUT ステートメントは修飾リスト入力を使用してデータレコードを正しく読み込みます。

```
data name;
  informat FirstName LastName $15. n1 6.2 n2 7.3;
  input firstname lastname n1 n2;
  datalines;
Alexander Robinson 35 11
;
```

```
proc contents data=name;
run;
proc print data=name;
run;
```

次の出力結果は、PROC CONTENTS の実行結果の一部と PROC PRINT で生成したレポートを示しています。

**アウトプット 2.9** SAS 変数への数値および文字の入力形式の関連付け

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Informat
1	FirstName	Char	15	\$15.
2	LastName	Char	15	\$15.
3	n1	Num	8	6.2
4	n2	Num	8	7.3

**アウトプット 2.10** PROC PRINT で生成したレポート

Obs	FirstName	LastName	n1	n2
1	Alexander	Robinson	0.35	0.011

### 例 3: 入力形式の取り消し

この例では既存の入力形式の関連付けを取り消します。INFORMAT ステートメントと SET ステートメントを指定する順序が重要です。

```
data rtest;
  set rtest;
  informat x;
run;
```

### 関連項目:

#### ステートメント:

- [“ATTRIB ステートメント” \(29 ページ\)](#)
- [“INPUT ステートメント” \(179 ページ\)](#)
- [“INPUT ステートメント、リスト” \(204 ページ\)](#)

---

## INPUT ステートメント

入力データレコードでの値の位置を指定し、読み込んだ値を対応する SAS 変数に割り当てます。

**該当要素:** DATA ステップ

**カテゴリ:** ファイル操作

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。

## 構文

```
INPUT <specification(s)> <@ | @@>;
```

### 引数なし

引数を指定しない INPUT ステートメントは、ヌル INPUT ステートメントと呼ばれます。ヌル INPUT ステートメントは、次の場合に使用します。

- SAS 変数を作成せずに、入力データレコードを入力バッファに読み込む場合
- 後置@または後置@@で保持されている入力データレコードを解放する場合

この例については、“例 2: [ヌル INPUT ステートメントを使用する](#)” (193 ページ)を参照してください。

## 引数

### *specification(s)*

次を指定できます。

#### *variable*

読み込む値を割り当てる変数名を 1 つ指定します。

#### *(variable-list)*

読み込む値を割り当てる変数名のリストを指定します。

要件 (variable-list)の後ろに(informat-list)を指定します。

参照項目: “[変数と入力形式をグループ化する方法](#)” (202 ページ)

#### \$

変数の値を数値ではなく文字値として格納するように指定します。

ヒント この変数が文字変数と事前に定義されている場合、\$を指定する必要はありません。

例 “[例 1: INPUT ステートメントで複数のスタイルを使用する](#)” (192 ページ)

### *pointer-control*

入力バッファ内の指定した行または列に入力ポインタを移動させます。

参照項目: “[カラムポインタコントロール](#)” (182 ページ) および “[行ポインタコントロール](#)” (184 ページ)

### *column-specifications*

読み込む値が含まれる入力レコードの列を指定します。

ヒント 入力形式は無視されます。この方法では、標準の文字データと数値データのみ正しく読み込めます。

参照項目: “[カラム入力](#)” (185 ページ)

例 “[例 1: INPUT ステートメントで複数のスタイルを使用する](#)” (192 ページ)



**format-modifier**

修飾リスト入力を使用できるようにします。また、入力値にエラーが発生したときに SAS ログに出力する情報の量を制御します。

**ヒント** 修飾リスト入力を使用すると、単純リスト入力では読み込めないデータを読み込めます。

**参照項目:** “リスト入力の使用が求められる場合” (207 ページ) および “エラーレポートのフォーマット修飾子” (184 ページ)

**例** “例 6: 文字変数でポインタの位置を指定する” (195 ページ)

**informat.**

変数の値の読み込みに使用する入力形式を指定します。

**ヒント** 修飾リスト入力を使用すると、入力形式を指定したデータを読み込めます。修飾リスト入力は、データに入力形式が必要ですが、値の列位置が揃っていないためフォーマット入力ではデータを読み込めない場合に使用すると便利です。

**参照項目:** “フォーマット入力” (186 ページ) および “リスト入力” (186 ページ)

**例** “例 2: 入力形式リストを使用する” (204 ページ)

**(informat-list)**

入力形式のリストを指定します。このリストは、前の位置に指定した変数リストの値の読み込みに使用します。

**制限事項** (*informat-list*)は(*variable-list*)の後ろに指定する必要があります。

**参照項目:** “変数と入力形式をグループ化する方法” (202 ページ)

**@**

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、**後置@**と呼ばれます。

**制限事項** 後置@は、INPUT ステートメントの最後の項目として指定する必要があります。

**ヒント** 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

**参照項目:** “ラインホールド指定子の使用” (188 ページ)

**例** “例 3: 入力バッファ内のレコードを保持する” (193 ページ)

**@@**

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、**後置@@**と呼ばれます。

制限事項	後置@@は、INPUT ステートメントの最後の項目として指定する必要があります。
ヒント	後置@@は、それぞれの入力行に複数のオブザベーションの値が含まれる場合や、DATA ステップの次の繰り返しでレコードをもう一度読み込む必要がある場合に便利です。
参照項目:	“ラインホールド指定子の使用”(188 ページ)
例	“例 4: DATA ステップの反復間を通してレコードを保持する”(194 ページ)

## カラムポインタコントロール

### @n

ポインタを列  $n$  に移動させます。

範囲 正の整数

ヒント  $n$  が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。 $n$  が 0 または負の場合、ポインタは列 1 に移動します。

例 @15 と指定すると、ポインタは列 15 に移動します。  
input @15 name \$10.;

例 “例 7: ポインタを後方(左)に移動する”(196 ページ)

### @numeric-variable

指定した *numeric-variable* の値が示す列にポインタを移動させます。

範囲 正の整数

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*numeric-variable* の値が 0 または負の整数の場合、ポインタを列 1 に移動させます。

例 変数 A の値に従って、ポインタは列 15 に移動します。  
a=15;  
input @a name \$10.;

例 “例 5: 数値変数でポインタの位置を指定する”(194 ページ)

### @(expression)

指定した *expression* の値が示す列にポインタを移動させます。

制限事項 *expression* の実行結果は正の整数でなければなりません。

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。値が 0 または負の整数の場合、ポインタを列 1 に移動させます。

例 式の実行結果に従って、ポインタを列 15 に移動します。  
b=5;  
input @(b\*3) name \$10.;

**@'character-string'**

指定した *character-string* を入力レコードから検索し、検出された文字列の終わりから 1 番目の列にポインタを移動させます。

**@character-variable**

指定した *character-variable* の値が示す文字列を入力レコードから検索し、検出された文字列の終わりから 1 番目の列にポインタを移動させます。

例 次のステートメントでは、文字変数 WEEKDAY を読み込みます。2 番目に指定した @1 に従って、ポインタを入力行の先頭に移動させます。変数 SALES の値は、変数 WEEKDAY の値の後方の非空白列から読み込まれます。

```
input @1 day 1. @5 weekday $10.
    @1 @weekday sales 8.2;
```

例 “例 6: 文字変数でポインタの位置を指定する” (195 ページ)

**@(character-expression)**

指定した *character-expression* の値が示す文字列を入力レコードから検索し、検出された文字列の終わりから 1 番目の列にポインタを移動させます。

例 “例 6: 文字変数でポインタの位置を指定する” (195 ページ)

**+n**

*n* に指定した列数だけポインタを移動させます。

範囲 正の整数または 0

ヒント *n* が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。この値が入力バッファの長さを超える場合、ポインタは次のレコードの列 1 に移動します。

例 次のステートメントでは、ポインタを列 23 に移動させます。LENGTH 変数の値として列 23 から 26 までを読み込みます。次に、ポインタを 5 列右に移動させ、WIDTH 変数の値として列 32 から 35 までを読み込みます。

```
input @23 length 4. +5 width 4.;
```

例 “例 7: ポインタを後方(左)に移動する” (196 ページ)

**+numeric-variable**

指定した *numeric-variable* の値が示す列数だけポインタを移動させます。

範囲 正の整数、負の整数、または 0

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*numeric-variable* の値が負の値の場合、ポインタを後方(左)へ移動させます。ただし、現在の列位置が 1 未満になる場合、ポインタは列 1 に移動します。値が 0 の場合、ポインタは移動しません。この値が入力バッファの長さを超える場合、ポインタは次のレコードの列 1 に移動します。

例 “例 7: ポインタを後方(左)に移動する” (196 ページ)

**+(expression)**

指定した *expression* の値が示す列数だけポインタを移動させます。

**範囲** *expression* の実行結果が、正の整数、負の整数または 0 になる式を指定します。

**ヒント** *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*expression* の実行結果が負の値の場合、ポインタを左に戻します。ただし、現在の列位置が 1 未満になる場合、ポインタは列 1 に移動します。値が 0 の場合、ポインタは移動しません。この値が入力バッファの長さを超える場合、ポインタは次のレコードの列 1 に移動します。

## 行ポインタコントロール

### #*n*

ポインタをレコード *n* に移動させます。

**範囲** 正の整数

**操作** INFILE ステートメントの N=オプションの値によって、INPUT ステートメントで読み込むレコード数や、DATA ステップの繰り返しを終了した後の入力ポインタの位置は異なります。N=(155 ページ)オプションの説明を参照してください。

**例** #2 と指定されているので、ポインタは 2 番目のレコードに移動してから、変数 ID の値として列 3 と 4 を読み込みます。  
input name \$10. #2 id 3-4;

### #*numeric-variable*

指定した *numeric-variable* の値が示すレコードにポインタを移動させます。

**範囲** 正の整数

**ヒント** *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

### #(*expression*)

指定した *expression* の値が示すレコードにポインタを移動させます。

**範囲** *expression* の実行結果は正の整数でなければなりません。

**ヒント** *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

### /

ポインタを次の入力レコードの列 1 に移動させます。

**例** 最初の入力レコードから変数 NAME と変数 AGE の値を読み込みます。次に、ポインタを 2 番目のレコードに移動し、変数 ID の値として列 3 と 4 を読み込みます。  
input name age / id 3-4;

## エラーレポートのフォーマット修飾子

### ?

無効なデータ値が検出されたときに、無効なデータに関するメッセージを出力しないようにします。

参照項目: [“無効なデータの処理方法” \(191 ページ\)](#)

??

無効なデータ値が検出されたときに、メッセージと該当する入力行を出力しないようにします。この無効なオブザベーションに対しては、自動変数 `_ERROR_` の値は 1 に設定されません。

参照項目: [“無効なデータの処理方法” \(191 ページ\)](#)

## 詳細

### INPUT を使用する必要がある場合

外部ファイルやインストリームデータから生データを読み込むには、INPUT ステートメントを使用します。データが外部ファイルに保存されている場合、INFILE ステートメントにそのファイルを指定できます。INFILE ステートメントは、データレコードを読み込む INPUT ステートメントよりも前に実行する必要があります。読み込むデータがインストリームデータの場合、ジョブストリーム内の DATALINES ステートメントに続けてデータ行を指定する必要があります。読み込むデータにセミコロンが含まれる場合、DATALINES4 ステートメントに続けてデータ行を指定します。生データを読み込む 1 つの DATA ステップには、複数の INPUT ステートメントを使用できます。

INPUT ステートメントの前に INFILE ステートメントの file-specification に DATALINES を指定すると、INFILE ステートメントを使用してインストリームデータを読み込むこともできます。INFILE ステートメントに DATALINES を指定すると、INFILE ステートメントのほとんどのオプションをインストリームデータの読み込みに使用できます。

SAS データセットに格納されているデータを読み込むには、SET ステートメントを使用します。SAS 以外のソフトウェアで作成されたデータベースまたは PC ファイル形式のデータを読み込むには、LIBNAME ステートメントでデータにアクセスした後、SET ステートメントを使用します。詳細については、SAS/ACCESS のドキュメントを参照してください。

## 入力スタイル

### 入力スタイルの概要

INPUT ステートメントでレコードの値を指定する方法は 4 つあります。

- カラム
- リスト(単純および修飾)
- フォーマット
- 名前付き

各変数の値は、次のいずれかの入力スタイルを使用して読み込まれます。INPUT ステートメントには、入力レコードでのデータ値の配置に応じて、使用可能な入力スタイルのいずれかまたはすべてを指定することができます。ただし、名前付き入力が入力スタイルで使用されると、他の入力スタイルを使用することはできません。

### カラム入力

カラム入力を使用する場合、INPUT ステートメントで、値の位置を示す列番号を変数名の後ろに指定します。番号は、入力データレコード内で変数の値が存在する位置を示しています。

```
input name $ 1-8 age 11-12;
```

この INPUT ステートメントを実行すると、次のデータレコードを読み込むことができます。

```
----+----1----+----2----+
Peterson 21
Morgan 17
```

NAME は文字変数なので、変数名と列番号の間に\$を指定しています。詳細については、「[INPUT ステートメント、カラム](#)」(197 ページ)を参照してください。

### リスト入力

リスト入力では、変数名を INPUT ステートメント内に単にリストします。文字変数名の後ろには\$を指定します。

```
input name $ age;
```

この INPUT ステートメントを実行すると、空白で区切られているデータ値、または列位置が揃っていないデータ値を読み込むことができます。ただし、それぞれの値の間には1つ以上の空白が必要です。

```
----+----1----+----2----+
Peterson 21
Morgan 17
```

詳細については、「[INPUT ステートメント、リスト](#)」(204 ページ)を参照してください。

### フォーマット入力

フォーマット入力では、INPUT ステートメントで変数名の後に入力形式を指定します。入力形式では、データの種類や入力値のフィールド長を指示します。入力形式を指定すると、パック 10 進などの非標準形式のデータや、カンマなどの特殊文字を含む数値を読み込むこともできます。

```
input name $char8. +2 income comma6.;
```

この INPUT ステートメントを実行すると、次のデータレコードが正しく読み込まれます。

```
----+----1----+----2----+
Peterson 21,000
Morgan 17,132
```

ポインタコントロールを+2と指定し、入力ポインタを変数 INCOME の値が含まれるフィールドに移動させています。詳細については、「[INPUT ステートメント、フォーマット](#)」(200 ページ)を参照してください。

### 名前付き入力

入力名前付き 名前付き入力 名前付き入力では、変数名の後に等号(=)を指定します。入力レコードから変数名と等号が検索されます。

```
input name= $ age=;
```

この INPUT ステートメントを実行すると、次のデータレコードが正しく読み込まれます。

```
----+----1----+----2----+
name=Peterson age=21
name=Morgan age=17
```

詳細については、「[INPUT ステートメント、名前付き](#)」(212 ページ)を参照してください。

## 1 つの INPUT ステートメントに複数のスタイルを指定する

1 つの INPUT ステートメントに異なる入力スタイルのいずれか、またはすべてを指定できます。

```
input idno name $18. team $ 25-30 startwght endwght;
```

この INPUT ステートメントを実行すると、次のデータレコードが正しく読み込まれます。

```
----+----1----+----2----+----3----+----
023 David Shaw    red  189 165
049 Amelia Serrano yellow 189 165
```

変数 IDNO、STARTWGHT、ENDWGHT の値はリスト入力で、変数 NAME の値はフォーマット入力で読み込まれます。また、変数 TEAM の値はカラム入力で読み込まれます。

注: INPUT ステートメントに名前付き入力を使用されると、入力スタイルを変更することはできません。

## ポインタコントロール

### ポインタの概要

入力データレコードから入力バッファに値を読み込む場合、ポインタを使用して読み込み位置を制御します。INPUT ステートメントには、ポインタの動きを制御する 3 つの方法があります。

#### カラムポインタコントロール

データレコードのデータ値を読み込むときに、ポインタの列位置をリセットします。

#### 行ポインタコントロール

データレコードのデータ値を読み込むときに、ポインタの行位置をリセットします。

#### ラインホールド指定子

読み込んだ入力レコードが入力バッファに保持されるので、他の INPUT ステートメントでも保持した値を使用できます。デフォルトでは、INPUT ステートメントを実行すると、前回読み込んだレコードを解放してから、次のレコードを読み込みます。

カラムポインタコントロールや行ポインタコントロールを使用すると、ポインタを移動させる行または列の絶対位置を指定したり、現在のポインタの位置の相対位置として移動先の列または行を指定できます。INPUT ステートメントで指定できるすべてのポインタコントロールを次の表に示します。

表 2.5 INPUT ステートメントで指定できるポインタコントロール

ポインタコントロール	相対指定	絶対指定
カラムポインタコントロール	$+n$	$@n$
	$+numeric-variable$	$@numeric-variable$
	$+(expression)$	$@(expression)$
		$@'character-string'$



ポインタコントロール	相対指定	絶対指定
		@character-variable
		@(character-expression)
行ポインタコントロール	/	#n
		#numeric-variable
		#(expression)
ラインホールド指定子	@	(なし)
	@@	(なし)

注: ポインタコントロールは、常に適用する変数の前に指定するようにしてください。

INFILE ステートメントに COLUMN=オプションや LINE=オプションを指定して、ポインタの現在の列位置や行の位置を確認することができます。

#### カラムポインタコントロールと行ポインタコントロールの使用

カラムポインタコントロールでは、入力値の読み込みを開始する列を指定します。

次の入力レコードに移動したり、オブザベーションごとの入力レコード数を定義するには、INPUT ステートメントで行ポインタコントロールを使用します。行ポインタコントロールには、読み込み対象とする入力レコードを指定します。複数のデータレコードを入力バッファに読み込むには、INFILE ステートメントの N=オプションに読み込むレコード数を指定します。N=オプションの指定を省略する場合は、特別な注意が必要になります。詳細については、「[1 つのオブザベーションにつき複数のレコードを読み込む](#)」(190 ページ)を参照してください。

#### ラインホールド指定子の使用

ラインホールド指定子を使用して、次の場合にポインタを現在の出力行に保持します。

- 複数の INPUT ステートメントで同じデータレコードを読み込む場合(後置@)
- 1 つの入力行に複数のオブザベーションの値が含まれる場合(後置@@)
- DATA ステップの次の反復でレコードをもう一度読み込む必要がある場合(後置@@)

後置@を使用すると、次の INPUT ステートメントで同じレコードを読み込むことができます。後置@@を使用すると、DATA ステップの反復間を通して、読み込んだレコードを保持して次の INPUT ステートメントで使用することができます。

通常、DATA ステップで INPUT ステートメントを実行するたびに、新しいデータレコードが入力バッファに読み込まれます。後置@を使用する場合は、次のようになります。

- ポインタの位置は変化しません。
- 入力バッファに新しいレコードは読み込まれません。



- DATA ステップの同じ反復内で次の INPUT ステートメントを実行すると、新しいレコードではなく同じレコードが読み込まれます。

後置@で保持したレコードは、次の場合に解放されます。

- ヌル INPUT ステートメントを実行した場合  
input;
- 後置@を指定せずに INPUT ステートメントを実行した場合
- DATA ステップの次の反復を開始した場合

通常、後置@@を使用すると、DATA ステップの次の反復でも INPUT ステートメントは同じレコードを読み込みます。後置@@で保持したレコードは、次の場合に解放されます。

- ポインタが入力レコードの行端を超えた場合
- ヌル INPUT ステートメントを実行した場合  
input;
- DATA ステップの次の反復が開始され、この DATA ステップで後置@を指定した INPUT ステートメントが実行された場合  
input @;

#### 値を読み込んだ後のポインタの位置

1つの INPUT ステートメントに複数の入力スタイルを組み合わせで使用する場合は、値を読み込んだ後の入力ポインタの位置を理解しておくことが重要になります。カラム入力とフォーマット入力を使用すると、ポインタは INPUT ステートメントに指定された列を読み込んだあと、次の列で停止します。リスト入力を使用すると、ポインタはデータレコードを走査してデータ値を探してから、値の終わりを示すブランクを読み込みます。そのため、リスト入力で値を読み込んだ後、ポインタは値の終わりから2番目の列で停止します。

たとえば、リスト入力、カラム入力、フォーマット入力を使用して次のデータレコードを読み込むことができます。

```
----+----1----+----2----+----3
REGION1  49670
REGION2  97540
REGION3  86342
```

次の INPUT ステートメントでは、リスト入力を使用してデータレコードを読み込みます。

```
input region $ jansales;
```

REGION の値を読み込んだ後、ポインタは列9で停止します。

```
----+----1----+----2----+----3
REGION1  49670
      ↑
```

次の INPUT ステートメントでは、カラム入力とフォーマット入力を使用してデータレコードを読み込みます。

- カラム入力  
input region \$ 1-7 jansales 12-16;
- フォーマット入力  
input region \$7. +4 jansales 5.;  
input region \$7. @12 jansales 5.;

変数 REGION の値を読み込むため、この INPUT ステートメントでは、7 列読み込んでから列 8 で停止するようにポインタに指示します。

```
----+----1----+----2----+----3
REGION1 49670
  ↑
```

## 1 つのオブザベーションにつき複数のレコードを読み込む

### #ポインタコントロールを使用する

INPUT ステートメントの#ポインタコントロールの後ろに指定した最大値によって、入力バッファに読み込む入力データレコード数が決まります。INFILE ステートメントに N=オプションを指定すると、このレコード数を変更することができます。たとえば、次のステートメントでは、#ポインタコントロールに指定した最大値は 3 になります。

```
input @31 age 3. #3 id 3-4 #2 @6 name $20.;
```

関連する INFILE ステートメントに N=オプションを指定していない場合、DATA ステップを実行するたびに、INPUT ステートメントは 3 つの入力レコードを読み込みます。

各オブザベーションに複数の入力レコードが存在しても、最後のレコードの値は読み込まない場合、INPUT ステートメントに#ポインタコントロールを使用するか、INFILE ステートメントに N=オプションを使用して最後に読み込む入力レコードを指定する必要があります。たとえば、各オブザベーションに 4 つのレコードが存在しても、最初の 2 つの入力レコードの値しか読み込まない場合、次の INPUT ステートメントを使用します。

```
input name $ 1-10 #2 age 13-14 #4;
```

/ポインタコントロールを指定してポインタを次のレコードに移動した場合、INPUT ステートメントに#ポインタコントロールを指定するか、INFILE ステートメントに N=オプションを指定して、入力バッファに読み込むレコード数を設定します。ポインタを前のレコードに戻すには、#ポインタコントロールを使用します。たとえば、次のステートメントでは、INFILE ステートメントに N=オプションを指定していない場合、#2 ポインタコントロールを指定してレコードを 2 つ読み込むように指示する必要があります。

```
input a / b #1 @52 c #2;
```

この INPUT ステートメントでは、変数 A に最初のレコードの値を割り当てます。ポインタを次の入力レコードに移動してから、変数 B に値を割り当てます。次に、ポインタを 2 番目のレコードから最初のレコードの列 1 に復帰した後に、列 52 に移動して変数 C に値を割り当てます。ポインタコントロールの#2 は、オブザベーションの入力レコードが 2 行にわたることを示すため、C の値を読み込むためにポインタが最初のレコードに復帰できます。

オブザベーションごとに入力レコード数が異なる場合、INFILE ステートメントの N=オプションにオブザベーションのレコード数の最大値を指定します。詳細については、[N=オプション \(155 ページ\)](#)を参照してください。

### 行の終わりを越えて読み込む

ポインタコントロールの@または+に指定した値が、現在のレコードの終端または終端を超えた位置に移動し、その列から次の値を読み込むように指示している場合、ポインタは、次のレコードの列 1 に移動してから値を読み込みます。次のメッセージが SAS ログに書き込まれます。

```
NOTE: SAS went to a new line when INPUT statement
       reached past the end of a line.
```

INFILE ステートメントでデフォルトの動作(FLOWOVER オプション)を変更することができます。

INFILE ステートメントに STOPOVER オプションを指定すると、この状態をエラーと判断し、データセットの作成を終了します。

INFILE ステートメントに MISCOVER オプションを指定すると、ポインタがレコードの終端に到達した場合、INPUT ステートメントでまだ値が読み込まれていない変数には欠損値を割り当てます。

INFILE ステートメントに TRUNCOVER オプションを指定すると、INPUT ステートメントで読み込んだ最後の変数に可変長データが含まれている場合、カラム入力またはフォーマット入力を使用します。

### レコードの前にポインタを移動する

カラムポインタコントロールをレコードの開始位置よりも前に移動するように指示した場合、ポインタは列 1 に移動します。たとえば、次の INPUT ステートメントでは、最初の値を読み込んだ後、ポインタを列-2 に移動するように指示しています。

```
data test;
  input a @(a-3) b;
  datalines;
2
;
```

そのため、変数 A の値を読み込んだ後、ポインタは列 1 に移動します。変数 A と B には同じ値が割り当てられます。

### 無効なデータの処理方法

指定した変数の入力値に無効な文字が検出された場合、次の処理を実行します。

- 読み込み中の変数の値を欠損値に設定するか、INVALIDDATA=システムオプションに指定した値に設定します。詳細については、[“INVALIDDATA= System Option” \(SAS Viya System Options: Reference\)](#)を参照してください。
- 無効なデータに関するメッセージが SAS ログに出力されます。
- 無効な値が含まれる入力行と列番号を SAS ログに出力します。表示不能な文字は、16 進表記で表示されます。カラム番号を見やすくするために、入力行の上部に罫線が出力されます。
- 現在のオブザベーションの自動変数\_ERROR\_の値が 1 に設定されます。

エラーレポートのフォーマット修飾子を指定すると、SAS ログに出力される情報の量を制御することができます。?および?? 修飾子を指定すると、無効なデータを示すメッセージを出力しないようにします。ただし、?? 修飾子では、自動変数\_ERROR\_の値が 0 に設定されます。たとえば、次の 2 つのステートメントは同じ結果になります。

- input x ?? 10-12;
- input x ? 10-12;
  - \_error\_=0;

どちらの場合でも、変数 X の値が無効な場合は欠損値に設定されます。

### ファイルの終端

INPUT ステートメントがデータを最後まで読み込むと、ファイルの終端が検出されます。DATA ステップでファイルの終端に到達した後も他のレコードを読み込もうとすると、DATA ステップの実行が終了します。DATA ステップの実行

を継続するには、INFILE ステートメントに END=オプションまたは EOF=オプションを使用します。その場合、ファイルの終端の検出時に、INPUT ステートメントの実行を停止しても DATA ステップの処理は継続するプログラムステートメントを作成することができます。詳細については、“[INFILE ステートメント](#)” (146 ページ)を参照してください。

## 配列

INPUT ステートメントでは、配列参照を使用して入力データ値を読み込むことができます。配列参照を丸かっこで囲むと、ポインタコントロールに配列参照を使用できます。“[例 6: 文字変数でポインタの位置を指定する](#)” (195 ページ)を参照してください。

配列の subscript にアスタリスク(\*)を使用すると、明示的に定義済みの配列のすべての要素を読み込むことができます。1次元配列または多次元配列を使用できます。subscript は中かっこ、大かっこ、丸かっこで囲んでください。このステートメントの形式は次のようになります。

```
INPUT array-name{*};
```

リスト入力、カラム入力、フォーマット入力で配列を使用できます。ただし、\_TEMPORARY\_ で定義されている配列や、subscript にアスタリスクを使用する配列に値を読み込むことはできません。たとえば、次のステートメントでは変数 X1 から X100 を作成し、入力形式 2.を使用してデータ値を変数に割り当てます。

```
array x{100};
input x{*} 2.;
```

## 比較

- INPUT ステートメントでは、外部ファイルの生データ、または DATALINE ステートメントの後ろに指定されているインストリームデータ行を読み込みます。これらは、SAS への記述が必要です。SET ステートメントでは、SAS データセットを読み込みます。このデータセットには、データ値の記述情報が含まれています。
- INPUT ステートメントはデータを読み込みます。PUT ステートメントはデータ値、テキスト文字列、またはその両方を SAS ログまたは外部ファイルに書き込みます。
- INPUT ステートメントは外部ファイルからデータを読み込みます。INFILE ステートメントは INPUT ステートメントで読み込むファイルや、ファイルを読み込む方法を制御するオプションを指定します。

## 例

### 例 1: INPUT ステートメントで複数のスタイルを使用する

この例では、1つの INPUT ステートメントで複数の入力スタイルを使用します。

```
data club1;
  input Idno Name $18.
        Team $ 25-30 Startwght Endwght;
  datalines;
023 David Shaw    red  189 165
049 Amelia Serrano  yellow 189 165
... more data lines ...
;
```

次の表に入力スタイルの種類を示します。

変数	入力の種類
Idno、Startwght、Endwght	リスト入力
Name	フォーマット入力
Team	カラム入力

### 例 2: ヌル INPUT ステートメントを使用する

この例では、引数を指定しない INPUT ステートメントを使用します。この DATA ステップを実行すると、SAS 変数を作成せずに、入力ファイルから出力ファイルにレコードがコピーされます。

```
data _null_;
  infile file-specification-1;
  file file-specification-2;
  input;
  put _infile_;
run;
```

### 例 3: 入力バッファ内のレコードを保持する

この例では、2 種類の入力データレコードを含むファイルを読み込み、読み込んだレコードから SAS データセットを作成します。1 つのデータレコードには、ある大学のコースに関する情報が保存されています。もう 1 つのデータレコードには、コースを専攻した学生の情報が保存されています。2 つのレコードを読み込み、出力形式が異なる複数の変数に値を割り当てるには、2 つの INPUT ステートメントを使用する必要があります。次に示すように、クラス情報を含むレコードの列 1 には C という値が格納されています。また、学生情報を含むレコードの列 1 には S という値が格納されています。

```
----+----1----+----2----+
C HIST101 Watson
S Williams 0459
S Flores 5423
C MATH202 Sen
S Lee 7085
```

どちらの INPUT ステートメントを使用するかを判断するため、このレコードの値を読み込むときに確認します。レコードに含まれる情報がクラスの情報か学生の情報かを判断する変数のみを読み込む INPUT ステートメントを使用します。

```
data schedule(drop=type);
  retain Course Professor;
  input type $1. @;
  if type='C' then
    input course $ professor $;
  else if type='S' then
    do;
      input Name $10. Id;
      output schedule;
    end;
  datalines;
C HIST101 Watson
S Williams 0459
```

```

S Flores 5423
C MATH202 Sen
S Lee 7085
;
run;

proc print;
run;

```

最初の INPUT ステートメントは、各行の列 1 から変数 TYPE の値を読み込みます。INPUT ステートメントの末尾に後置@が指定されているので、DATA ステップの次の INPUT ステートメントでも同じ行を読み込みます。最初の INPUT ステートメントの後に記述されている IF-THEN ステートメントは、次の INPUT ステートメントで行の残りを読み込む前に、レコードに含まれる情報がクラスの情報が学生の情報かを判断します。後置@を指定されていないこの INPUT ステートメントは、現在保持している行を解放します。RETAIN ステートメントでは、大学のコースの値が保持されます。この DATA ステップは、学生情報のレコードを読み込んだ後、オブザベーションを SCHEDULE データセットに書き込みます。

生成されたデータセット SCHEDULE の内容を PROC PRINT で出力した結果を次に示します。

#### アウトプット 2.11 データセット SCHEDULE

Obs	Course	Professor	Name	Id
1	HIST101	Watson	Williams	459
2	HIST101	Watson	Flores	5423
3	MATH202	Sen	Lee	7085

#### 例 4: DATA ステップの反復間を通してレコードを保持する

この例は、1つの入力データレコードから複数のオブザベーションを作成する方法を示しています。各レコードには、変数 NAME と変数 AGE の値が複数含まれています。この DATA ステップでは、変数 NAME の値と変数 AGE の値をそれぞれ 1 つ読み込んでからオブザベーションを 1 つ出力します。レコード内のすべての入力値が処理されるまで、NAME と AGE の値の組み合わせを継続して読み込みます。

```

data test;
  input name $ age @@;
  datalines;
John 13 Monica 12 Sue 15 Stephen 10
Marc 22 Lily 17
;

```

この INPUT ステートメントでは後置@@が指定されているので、DATA ステップの反復間で入力ポインタが制御されます。生成される SAS データセットには、6 つのオブザベーションが含まれます。

#### 例 5: 数値変数でポインタの位置を指定する

この例では、数値変数を使用してポインタの位置を指定します。生データファイルには、多国籍企業の複数の事業所別の従業員数が保存されています。入力データレコードは次のとおりです。

```

-----1-----2-----3-----
8 New York 1 USA 14
5 Cary      1 USA 2274
3 Chicago   1 USA 37
22 Tokyo    5 ASIA 80
5 Vancouver 2 CANADA 6
9 Milano    4 EUROPE 123

```

最初の列は、事業所の所在地名の開始列位置を含みます。次の数値列は、地域カテゴリを示しています。また、各事業所の従業員数の前に地域名が示されています。

`@numeric-variable` ポインタコントロールと後置`@`を組み合わせ使用し、事業所の所在地名の開始列位置を指定します。レコードを読み込みには、INPUT ステートメントを2つ使用します。最初の INPUT ステートメントでは、`@numeric-variable` ポインタコントロールの値を取得します。2番目の INPUT ステートメントでは取得した値を使用して、ポインタの移動先となる列を指示します。

```

data office (drop=x);
  infile file-specification;
  input x @;
  if 1<=x<=10 then
    input @x City $9.;
  else do;
    put 'Invalid input at line ' _n_;
    delete;
  end;
run;

```

この DATA ステップでは、OFFICE データセットに5つのオブザベーションのみが書き込まれます。4番目の入力データレコードは、変数 X の値が10を超えているので無効になります。そのため、2番目の INPUT ステートメントは実行されません。そのかわり、PUT ステートメントによってメッセージが SAS ログに書き込まれます。また、DELETE ステートメントによってオブザベーションの処理が中止されます。

### 例 6: 文字変数でポインタの位置を指定する

この例では、文字変数を使用してポインタの位置を指定します。“例 5: 数値変数でポインタの位置を指定する” (194 ページ) で作成された OFFICE データセットには、文字変数 CITY が含まれています。この変数の値は、事業所の所在地を示しています。ここで、生データファイルから、他の値を読み込む必要が発生したと仮定します。もう1つ DATA ステップを追加することで、その値を検出するために、`@character-variable` ポインタコントロール、後置`@`、`@character-expression` ポインタコントロールを組み合わせられます。

OFFICE データセットに含まれるオブザベーションが元の入力データレコードと同じ順で格納されている場合、次の DATA ステップを使用することができます。

```

data office2;
  set office;
  infile file-specification;
  array region {5} $ _temporary_
    ('USA' 'CANADA' 'SA' 'EUROPE' 'ASIA');
  input @city Location : 2. @;
  input @(trim(region{location})) Population : 4.;
run;

```



ARRAY ステートメントは、一時配列要素に初期値を割り当てます。この要素は、事業所の所在地の地域に対応しています。最初の INPUT ステートメントには、@*character-variable* ポインタコントロールが指定されています。各レコードから変数 CITY の値を示す文字列を走査します。次に、LOCATION の値を変数 CITY の次にある非空白列から読み込みます。LOCATION は、事業所の地域を示す数値になります。2 番目の INPUT ステートメントでは、@*character-expression* ポインタコントロールに配列参照を使用して、入力レコード内の変数 POPULATION の位置を指示します。また、この式では、TRIM 関数を使用して、文字値の末尾にある空白を取り除きます。このようにして、入力データの文字列と配列要素の値の完全一致を検出します。

生成されたデータセット OFFICE2 の内容を PROC PRINT で出力した結果を次に示します。

アウトプット 2.12 データセット OFFICE2

Obs	City	Location	Population
1	New York	1	14
2	Cary	1	2274
3	Chicago	1	37
4	Tokyo	5	80
5	Vancouver	2	6
6	Milano	4	123

### 例 7: ポインタを後方(左)に移動する

この例では、ポインタを戻すいくつかの方法を示します。

- 次の INPUT ステートメントでは、@ポインタコントロールを使用して列 26 から始まる変数 BOOK の値を読み込みます。次に、ポインタを同じ行の列 1 に戻し、変数 COMPANY の値を読み込みます。

```
input @26 book $ @1 company;
```

- 次の INPUT ステートメントでは、+*numeric-variable* または+(*expression*)を使用し、ポインタを 1 列戻します。次の 2 つのステートメントは同じ結果になります。

- m=-1;  
input x 1-10 +m y 2.;
- input x 1-10 +(-1) y 2.;

### 関連項目:

#### ステートメント:

- [“ARRAY ステートメント” \(20 ページ\)](#)
- [“INPUT ステートメント、カラム” \(197 ページ\)](#)
- [“INPUT ステートメント、フォーマット” \(200 ページ\)](#)



- [“INPUT ステートメント、リスト” \(204 ページ\)](#)
- [“INPUT ステートメント、名前付き” \(212 ページ\)](#)

---

## INPUT ステートメント、カラム

指定した列から入力値を読み込み、読み込んだ値を対応する SAS 変数に割り当てます。

<b>該当要素:</b>	DATA ステップ
<b>カテゴリ:</b>	ファイル操作
<b>種類:</b>	実行
<b>制限事項:</b>	このステートメントは CAS Server ではサポートされていません。

---

### 構文

```
INPUT variable <$> start-column <- end-column>
<.decimals> <@ | @@>;
```

### 引数

#### *variable*

読み込む値を割り当てる変数を 1 つ指定します。

#### \$

変数の値が数値ではなく文字列であることを指定します。

**ヒント** この変数が文字変数と事前に定義されている場合、\$を指定する必要はありません。

#### *start-column*

読み込む値が含まれる入力レコードの開始列を指定します。

#### *-end-column*

読み込む値が含まれる入力レコードの最終列を指定します。

**ヒント** 変数の値が 1 列だけの場合、*end-column* の指定を省略することができます。

**例** *end-column* が省略されているため、文字変数 GENDER の値は列 16 にのみ存在します。

```
input name $ 1-10 pulse 11-13 waist 14-15 gender $ 16;
```

#### *.decimals*

入力値に小数点が明示的に含まれていない場合、小数点以下の桁数を指定します。

**ヒント** 入力値に小数点が明示的に含まれている場合、INPUT ステートメントで指定した小数点以下の桁数より優先されます。

**例** [“例 2: 小数点以下の桁数を指定した入力データの読み込み” \(199 ページ\)](#)

---

**@**

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、**後置@**と呼ばれます。

**制限事項** 後置@は、INPUT ステートメントの最後の項目として指定する必要があります。

**ヒント** 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

**参照項目:** [“ポインタコントロール” \(187 ページ\)](#)

**@@**

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、**後置@@**と呼ばれます。

**制限事項** 後置@@は、INPUT ステートメントの最後の項目として指定する必要があります。

**ヒント** 後置@@は、入力行に複数のオブザベーションの値が含まれる場合に便利です。

**参照項目:** [“ラインホールド指定子の使用” \(188 ページ\)](#)

**詳細****カラム入力の使用が求められる場合**

カラム入力を使用する場合、INPUT ステートメントで、値の位置を示す列番号を変数名の後ろに指定します。カラム入力でデータを読み込む場合、データ値は次の条件を満たす必要があります。

- 読み込むデータがすべての入力データレコードで同じ列位置に存在する
- 標準的な数値形式または文字形式

カラム入力の便利な機能を次に示します。

- 文字値の中に埋め込みブランクを含めることができます。
- 文字値の長さは、1 - 32,767 文字です。
- 入力値がレコードのどの位置にあっても、任意の順序で読み込むことができます。
- 値または値の一部を繰り返し読み込むことができます。たとえば、次の INPUT ステートメントでは、ID の値として列 10 から 15 までを読み込んだ後、GROUP の値として列 13 を読み込みます。

```
input id 10-15 group 13;
```

- フィールド内の前置ブランクと後置ブランクは無視されます。そのため、数値に 0 を示すブランクが含まれていたり、文字値の先頭と末尾にあるブランクを保持する場合は、入力形式を使用して値を読み込みます。[“INPUT ステートメント、フォーマット” \(200 ページ\)](#)を参照してください。

## 欠損値

欠損データにプレースホルダは必要ありません。INPUT ステートメントでは、空白のフィールドを欠損値として読み込みます。また、それ以外の値は正確に読み込みます。数値または文字列のフィールドにピリオドが1つだけ含まれている場合、この変数の値は欠損値に設定されます。

## データ行の読み込み

DATALINES ステートメントの後ろに指定したデータレコード(インストリームデータ)には、80の倍数を示す固定長になるように空白が追加されます。80番目以降の列のデータを読み込むか切り捨てるかは、CARDIMAGE システムオプションの設定によって異なります。

## 可変長レコードの読み込み

デフォルトでは、可変長データレコードの読み込みには、FLOWOVER オプションを使用します。レコードに含まれる値の数が必要な数よりも少ない場合、INPUT ステートメントは次のデータレコードから値を読み込みます。可変長データを読み込むには、INFILE ステートメントに TRUNCOVER オプションを指定することをお勧めします。TRUNCOVER オプションは、レコードに空白を追加して固定長レコードに変更する PAD オプションよりも便利です。詳細については、“[行の終わりを越えて読み込む](#)” (165 ページ)を参照してください。

## 例

### 例 1: カラム入力での入力レコードの読み込み

次の DATA ステップは、カラム入力での入力データレコードを読み込む方法を示しています。

```
data scores;
  input name $ 1-18 score1 25-27 score2 30-32
        score3 35-37;
  datalines;
Joseph      11  32  76
Mitchel     13  29  82
Sue Ellen   14  27  74
;
```

### 例 2: 小数点以下の桁数を指定した入力データの読み込み

次の INPUT ステートメントでは、小数点以下2桁と指定して数値変数の入力データを読み込みます。

入力データ	ステートメント	結果
----+---1		
2314	input number 1-5 .2;	23.14
2		.02
400		4.00
-140		-1.40

入力データ	ステートメント	結果
12.234		12.234 *
12.2		12.2 *

\* INPUT ステートメントに指定した小数点以下の桁数は、入力値に小数点が含まれている場合は無効になります。

## 関連項目:

### ステートメント:

- [“INPUT ステートメント” \(179 ページ\)](#)

---

## INPUT ステートメント、フォーマット

指定した形式で入力値を読み込み、読み込んだ値を対応する SAS 変数に割り当てます。

**該当要素:** DATA ステップ

**カテゴリ:** ファイル操作

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。

## 構文

**INPUT** *<pointer-control>* *variable informat.* *<@ | @@>*;

**INPUT** *<pointer-control>* (*variable-list*) (*informat-list*)  
*<@ | @@>*;

**INPUT** *<pointer-control>* (*variable-list*) (*<n\*> informat.*)  
*<@ | @@>*;

## 引数

### **pointer-control**

入力バッファ内の指定した行または列に入力ポインタを移動させます。

参照項目: [“コラムポインタコントロール” \(182 ページ\)](#) および [“行ポインタコントロール” \(184 ページ\)](#)

### **variable**

読み込む値を割り当てる変数を 1 つ指定します。

要件 (*variable-list*)の後ろに(*informat-list*)を指定します。

例 [“例 1: フォーマット入力でポインタコントロールを使用する” \(203 ページ\)](#)

---

**(variable-list)**

読み込む値を割り当てる変数名のリストを指定します。

参照項目: “変数と入力形式をグループ化する方法” (202 ページ)

例 “例 2: 入力形式リストを使用する” (204 ページ)

**informat.**

変数の値の読み込みに使用する SAS 入力形式を指定します。

ヒント 実際の入力値に含まれる小数点は、数値入力形式による小数点の指定よりも優先されます。

参照項目: [SAS Viya Formats and Informats: Reference](#) の SAS 入力形式目:

例 “例 1: フォーマット入力でポインタコントロールを使用する” (203 ページ)

**(informat-list)**

入力形式のリストを指定します。このリストは、前の位置に指定した変数リストの値の読み込みに使用します。

INPUT ステートメントでは、(informat-list)に次を指定できます。

**informat.**

変数の値の読み込みに使用する入力形式を指定します。

**pointer-control**

値の読み込み位置を指示するポインタコントロール(@、#、/、+)のうちの1つを指定します。

**n\***

入力形式リストで n\*に続けて指定した入力形式を n 回繰り返します。

例 次のステートメントは、7.2 入力形式を使用して GRADES1、GRADES2、GRADES3 を読み込み、5.2 入力形式を使用して GRADES4、GRADES5 を読み込みます。

```
input (grades1-grades5)(3*7.2, 2*5.2);
```

制限事項 (informat-list)は(variable-list)の後ろに指定する必要があります。

参照項目: “変数と入力形式をグループ化する方法” (202 ページ)

例 “例 2: 入力形式リストを使用する” (204 ページ)

**@**

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、後置@と呼ばれます。

制限事項 後置@は、INPUT ステートメントの最後の項目として指定する必要があります。

ヒント 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

参照項目 “ラインホールド指定子の使用” (188 ページ)  
目:

### @@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、後置@@と呼ばれます。

制限事項 後置@@は、INPUT ステートメントの最後の項目として指定する必要があります。

ヒント 後置@@は、入力行に複数のオブザベーションの値が含まれる場合に便利です。

参照項目: “ラインホールド指定子の使用” (188 ページ)

## 詳細

### フォーマット入力の使用が求められる場合

フォーマット入力を使用する場合、変数名の後ろに入力形式を指定することにより、変数の値を読み込む方法を定義します。入力形式では、データの種類や入力値のフィールド長を指示します。入力形式を指定すると、パック 10 進などの非標準形式のデータや、カンマなどの特殊文字を含む数値を読み込むこともできます。SAS 入力形式については、“[Definition of Informats](#)” (*SAS Viya Formats and Informats: Reference*) を参照してください。

通常のフォーマット入力を使用する場合、対応する値が入力データに表示されるのと同じ順序で変数を指定する必要があります。ポインタコントロールを使用すると、変数を任意の順序で読み込むことができます。詳細については、“[INPUT ステートメント](#)” (179 ページ) を参照してください。

### 欠損値

通常、フォーマット入力では、数値の欠損値は 1 つのピリオドで示されます。また、文字の欠損値は 1 つの空白で示されます。フォーマット入力で空白がどのように解釈されるかは、使用する入力形式によって異なります。たとえば、\$CHAR.w と指定すると、空白は値の一部として読み込まれます。ただし、BZ.w と指定すると、空白は 0 に変換されます。

### 可変長レコードの読み込み

デフォルトでは、可変長データレコードの読み込みには、FLOWOVER オプションを使用します。レコードに含まれる値の数が必要な数よりも少ない場合、INPUT ステートメントは次のデータレコードから値を読み込みます。可変長データを読み込むには、INFILE ステートメントで TRUNCCOVER オプションを指定することをお勧めします。詳細については、“[行の終わりを越えて読み込む](#)” (165 ページ) を参照してください。

### 変数と入力形式をグループ化する方法

読み込む値があるパターンで配置されている場合、入力形式リストをグループ化することができます。グループ化した入力形式リストは、次の 2 つのリストで構成されます。

- 読み込み対象の変数名を丸かっこで囲んで指定したリスト
- ブランク区切りまたはカンマ区切りの対応する入力形式を丸かっこで囲んで指定したリスト

入力形式リストを使用すると INPUT ステートメントを簡潔に記述できます。これは、入力形式リストはすべての変数が読み込まれるまで繰り返し適用され、簡略化された番号付きの変数名が使用されるためです。入力形式リストを使用すると、個々の変数を指定する必要がなくなります。

たとえば、SCORE1 から SCORE5 までの 5 つの変数の値が、値ごとに 4 つの列として間の空白なしで格納されている場合、次の INPUT ステートメントはこれらの値を読み込みます。

```
input (score1-score5) (4. 4. 4. 4.);
```

ただし、入力形式よりも多くの変数を指定した場合、INPUT ステートメントは、入力形式のリストを再利用することで残りの変数を読み込みます。前述のステートメントは次のように簡略化できます。

```
input (score1-score5) (4.);
```

INPUT ステートメントには必要な数だけ入力形式リストを使用できますが、入力形式リストはネストさせないでください。変数リストにあるすべての値が読み込まれると、INPUT ステートメントは入力形式リストに残っている指示を無視します。この例については、“[例 3: 必要な数以上の入力形式が指定されている場合](#)” (204 ページ)を参照してください。

入力形式リストに  $n^*$  修飾子を使用すると、後ろに指定した入力形式を  $n$  回繰り返します。次に例を示します。

```
input (name score1-score5) ($10. 5*4.);
```

## 入力形式の保存方法

INPUT に指定した入力形式は、SAS データセットに保存されません。INFORMAT ステートメントや ATTRIB ステートメントに指定した入力形式は恒久的に保存されます。そのため、後続の DATA ステップでは、恒久的に保存された入力形式を使用してデータ値を読み込むことができます。入力形式を指定したり、PROC FSEDIT を使用して正しい形式でデータを入力する必要はありません。

## 比較

フォーマット入力に変数を読み込むときのポインタの動きは、カラム入力のポインタの動きと似ています。ポインタは入力形式に指定された長さを移動した後、次の列で停止します。列位置が揃っていないデータを入力形式を使用して読み込むには、[修飾リスト入力](#)を使用します。修飾リスト入力を使用すると、リスト入力の走査機能を使用できます。“[リスト入力の使用が求められる場合](#)” (207 ページ)を参照してください。

## 例

### 例 1: フォーマット入力でのポインタコントロールを使用する

次の INPUT ステートメントは、入力形式とポインタコントロールを使用します。

```
data sales;
  infile file-specification;
  input item $10. +5 jan comma5. +5 feb comma5.
    +5 mar comma5.;
run;
```

このステートメントでは、次の入力データレコードが読み込まれます。

```
----+----1----+----2----+----3----+----4
```



```
trucks    1,382  2,789  3,556
vans      1,265  2,543  3,987
sedans    2,391  3,011  3,658
```

ITEM の値は、レコードの最初の 10 列から読み込まれます。ポインタは 11 列の位置で停止します。末尾にあるブランクを取り除いてから、ITEM の値がプログラムデータベクトルに書き込まれます。次に、ポインタを 5 列右に移動してから、INPUT ステートメントで COMMA5.入力形式を使用して JAN の値を読み込みます。この入力形式は、カンマを含む数値の読み込みを実行し、フィールド長として 5 列使用します。その後も、ポインタを 5 列ずつ右に移動してから、INPUT ステートメントで COMMA5.入力形式を使用して FEB と MAR の値を読み込みます。

### 例 2: 入力形式リストを使用する

次の INPUT ステートメントは、文字入力形式 \$10. で変数 NAME の値を読み込みます。また、数値入力形式 4. で SCORE1 から SCORE5 までの変数の値を読み込みます。

```
data scores;
  input (name score1-score5) ($10. 5*4.);
  datalines;
Whittaker 121 114 137 156 142
Smythe    111 97  122 143 127
;
```

### 例 3: 必要な数以上の入力形式が指定されている場合

次の入力形式リストには、INPUT ステートメント実行時に必要となる数よりも多い入力形式が指定されています。

```
data test;
  input (x y z) (2.,+1);
  datalines;
2 24 36
0 20 30
;
```

この INPUT ステートメントでは、入力形式 2. を使用して X の値を読み込みます。カラムポインタコントロールは +1 と指定されているため、ポインタを 1 列だけ前方(右)に移動します。次に、入力形式 2. を使用して Y の値を読み込みます。カラムポインタコントロールは +1 と指定されているため、再度ポインタを 1 列だけ前方(右)に移動します。最後に、入力形式 2. を使用して Z の値を読み込みます。3 回目の繰り返しでは、INPUT ステートメントはポインタコントロールの移動を指示する +1 を無視します。

### 関連項目:

#### ステートメント:

- [“INPUT ステートメント” \(179 ページ\)](#)
- [“INPUT ステートメント、リスト” \(204 ページ\)](#)

---

## INPUT ステートメント、リスト

入力データレコードの入力値を走査し、対応する SAS 変数に割り当てます。



該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	実行
制限事項:	このステートメントは CAS Server ではサポートされていません。

## 構文

**INPUT** <pointer-control> variable <\$> <&> <@ | @@>;

**INPUT** <pointer-control> variable <: | & | ~>  
<informat.> <@ | @@>;

## 引数

### **pointer-control**

入力バッファ内の指定した行または列に入力ポインタを移動させます。

参照項目: “カラムポインタコントロール” (182 ページ) および “行ポインタコントロール” (184 ページ)

例: “例 2: ブランクが埋め込まれた文字データの読み込み” (210 ページ)

### **variable**

読み込む値を割り当てる変数を 1 つ指定します。

#### **\$**

変数の値を数値ではなく文字値として格納するように指定します。

ヒント: この変数が文字変数と事前に定義されている場合、\$を指定する必要はありません。

例: “例 1: 単純リスト入力を使用した位置が揃っていないデータの読み込み” (209 ページ)

#### **&**

文字値に 1 つ以上の空白が含まれている場合に指定します。このフォーマット修飾子を指定すると、次の非空白列から値を、連続する 2 つの空白、変数に定義された長さ、入力行の最後のいずれかにポインタが到達するまで読み込みます。

制限事項: & 修飾子は適用対象となる変数名と \$ 記号の後ろに指定する必要があります。

ヒント: & 修飾子の後ろに入力形式を指定しても、連続する 2 つの空白というフォーマット修飾子の終了条件は変わりません。

参照項目: “修飾リスト入力” (208 ページ)

例: “例 2: ブランクが埋め込まれた文字データの読み込み” (210 ページ)

#### **:**

INPUT ステートメントで変数の値の読み込みに使用する入力形式を指定できます。文字変数の場合、このフォーマット修飾子を指定すると、次の非空白の列から値を、次の空白の列、長さを指定した変数、データ行の

最後のいずれかにポインタが到達するまで読み込みます。数値変数の場合、このフォーマット修飾子を指定すると、次の非空白列の列から値を、次の空白の列、データ行の最後のいずれかにポインタが到達するまで読み込みます。

**ヒント** 変数の長さが定義されていない場合、入力形式の長さに従って値を読み込んでから格納されます。

ポインタは、次の空白列に到達するまで読み込みを継続します。ただし、フィールドの長さが入力形式の長さよりも長い場合、入力形式の長さまで切り捨てられます。

**参照項目:** “修飾リスト入力” (208 ページ)

**例** “例 3: 入力形式を使用した位置が揃っていないデータの読み込み” (210 ページ)

“例 5: 修飾リスト入力を使用した区切られたデータの読み込み” (211 ページ)

~

一重引用符、二重引用符、区切り文字を特別な方法で扱うように指示します。このフォーマット修飾子を指定すると、区切り文字ではなく文字として、引用符で囲まれた文字値内の区切り文字を読み込みます。また、変数に値を書き込む際、この引用符を保持します。

**制限事項** INFILE ステートメントに DSD オプションを使用する必要があります。使用しない場合、INPUT ステートメントはこのオプションは無視します。

**参照項目:** “修飾リスト入力” (208 ページ)

**例** “例 5: 修飾リスト入力を使用した区切られたデータの読み込み” (211 ページ)

### ***informat.***

変数の値の読み込みに使用する入力形式を指定します。

**ヒント** 実際の入力値に含まれる小数点は、数値入力形式による小数点の指定より優先されます。

**参照項目:** SAS Viya Formats and Informats: Reference の SAS 入力形式

**例** “例 3: 入力形式を使用した位置が揃っていないデータの読み込み” (210 ページ)

“例 5: 修飾リスト入力を使用した区切られたデータの読み込み” (211 ページ)

@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、後置@と呼ばれます。

**制限事項** 後置@は、INPUT ステートメントの最後の項目として指定する必要があります。

**ヒント** 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

**参照項目:** [“ラインホールド指定子の使用” \(188 ページ\)](#)

## @@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、後置@@と呼ばれます。

**制限事項** 後置@@は、INPUT ステートメントの最後の項目として指定する必要があります。

**ヒント** 後置@@は、入力行に複数のオブザベーションの値が含まれる場合に便利です。

**参照項目:** [“ラインホールド指定子の使用” \(188 ページ\)](#)

## 詳細

### リスト入力の使用が求められる場合

リスト入力では、入力データレコードでフィールドが表示されるのと同じ順序で、INPUT ステートメントに変数名を指定する必要があります。SAS ではデータ行を走査して次の値を検出しますが、値を区切る空白は無視されます。リスト入力では、データが特定の列位置に存在していなくてもかまいません。ただし、区切り文字を変更しないかぎり、値と値の間は1つ以上の空白で区切る必要があります。デフォルトでは、データ値の区切り文字は1つの空白、または入力レコードの最後に設定されています。リスト入力では、あるデータをスキップして後続のデータを読み込むことはできませんが、データレコード内の指定した位置より後ろにあるすべてのデータを無視することはできます。ただし、ポインタコントロールを使用すると、データ値を読み込む順序を変更することができます。

リスト入力には、次の2種類あります。

- 単純リスト入力
- 修飾リスト入力

フォーマット修飾子を使用すると単純リスト入力の複数の制限を取り除くことができるため、修飾リスト入力により INPUT ステートメントを幅広い用途に使用できるようになります。[“修飾リスト入力” \(208 ページ\)](#)を参照してください。

### 単純リスト入力

単純リスト入力では、INPUT ステートメントで読み込めるデータの種類の制限があります。

- デフォルトでは、入力値は1つ以上の空白で区切る必要があります。空白以外の区切り文字を使用する場合は、INFILE ステートメントに DLM=

オプション、DLMSTR=オプション、DSD オプションのいずれかを使用します。

- 欠損値は、ブランクや連続する 2 つの区切り文字ではなく、ピリオドで表示します。
- 文字入力値には 8 バイトを超えた値を使用できません。ただし、あらかじめ LENGTH、ATTRIB、INFORMAT の各ステートメントでそれ以上の変数の長さ指定している場合を除きます。
- 区切り文字を変更しない場合、文字値にブランクを使用することはできません。
- 標準の数値形式または文字形式のデータでなければなりません。

### 修飾リスト入力

フォーマット修飾子を使用すると、リスト入力をより多目的に使えるようになります。フォーマット修飾子を次に示します。

フォーマット修飾子	用途
&	埋め込みのブランクを含む文字値を読み込みます。
で定義された値は保持されません。	入力形式が提供する追加指示を必要とする、列位置が不揃いのデータ値を読み込みます。*
~	引用符で囲んだ文字値内の区切り文字を通常の文字として読み込みます。引用符は保持されます。
* フォーマット入力とポインタコントロールを使用して、列位置が揃っているデータ値を読み込みます。	

たとえば、:(コロン)修飾子と入力形式を併用すると、8 バイトを超える文字値や非標準の値を含む数値を読み込むことができます。

リスト入力ではブランクが区切り文字として解釈されるので、ブランクを含む値を読み込むには、修飾リスト入力を使用します。&修飾子を使用すると、連続していないブランクを含む文字値を読み込みます。ただし、データ値は 2 つ以上のブランクで区切られている必要があります。リスト入力を使用して先頭、末尾、内部のどこかにブランクを含むデータを読み込むには、INFILE ステートメントに DLM=オプションまたは DLMSTR=オプションを使用して他の文字を区切り文字に指定します。“[例 5: 修飾リスト入力を使用した区切られたデータの読み込み](#)” (211 ページ)を参照してください。区切り文字がブランクで、先頭、末尾、内部のどこかにブランクを含む入力データには、カラム入力またはフォーマット入力のいずれかを使用することをお勧めします。引用符で区切値が囲まれている場合、INFILE ステートメントに DSD オプションを指定すると、リスト入力を使用できます。

## 比較

### 修飾リスト入力とフォーマット入力の違い

修飾リスト入力には走査機能があるため、入力形式を使用して、列位置が不揃いのデータを読み込みます。フォーマット入力では、カラム入力と同じようにポインタを移動させて変数の値を読み込みます。ポインタは入力形式に指定された長さを移動した後、次の列で停止します。

次の DATA ステップでは、修飾リスト入力で最初のデータを読み込み、フォーマット入力をで 2 番目のデータを読み込みます。

```
data jansales;
  input item : $10. amount comma5.;
datalines;
trucks 1,382
vans 1,235
sedans 2,391
;
```

ITEM は修飾リスト入力で読み込まれます。INPUT ステートメントは、ポインタがブランクに到達した時点で読み込みを停止します。ポインタは、このフィールドの末尾から 2 列先(右)に移動します。この位置からフォーマット入力で AMOUT の値を読み込みます。

一方、フォーマット入力は、フィールドの総幅を読み込むまで読み込みを継続します。次の INPUT ステートメントでは、フォーマット入力を使用して両方のデータ値を読み込みます。

```
input item $10. +1 amount comma5.;
```

フォーマット入力でデータを正しく読み込むには、次に示すように 2 番目のデータ値が最初の値から 10 列目に出現する必要があります。

```
----+----1----+----2
trucks  1,382
vans    1,235
sedans  2,391
```

また、フォーマット入力で ITEM の値を読み込んだ後、ポインタコントロールを +1 と指定し、AMOUNT の値の開始位置となる列にポインタを移動する必要があります。

### データに引用符が含まれている場合

INFILE ステートメントに DSD オプションを使用して区切り文字をカンマに変更した場合、値を変数に書き込む前にデータに含まれる引用符が取り除かれます。また、INPUT ステートメントでチルダ(~)修飾子を使用すると、引用符を値の一部として保持することができます。

## 例

### 例 1: 単純リスト入力を使用した位置が揃っていないデータの読み込み

次の DATA ステップの INPUT ステートメントは、単純リスト入力を使用して入力データレコードを読み込みます。

```
data scores;
  input name $ score1 score2 score3 team $;
datalines;
Joe 11 32 76 red
Mitchel 13 29 82 blue
Susan 14 27 74 green
;
```

次の INPUT ステートメントは、前述のデータ行にある最初の 4 つのフィールドだけを読み込みます。この例で示すように、レコード内のすべてのデータを読み込む必要はありません。

```
input name $ score1 score2 score3;
```

**例 2: ブランクが埋め込まれた文字データの読み込み**

次の DATA ステップの INPUT ステートメントでは、& フォーマット修飾子とリスト入力を組み合わせて、ブランクを含む文字値を読み込みます。

```
data list;
  infile file-specification;
  input name $ & score;
run;
```

このステートメントでは、次の入力データレコードが読み込まれます。

```
----+----1-----+----2-----+----3-----+
Joseph 11 Joergensen red
Mitchel 13 Mc Allister blue
Su Ellen 14 Fischer-Simon green
```

INPUT ステートメントに適用する&修飾子が変数の後ろに指定されています。このフォーマット修飾子は NAME の後ろにあるので、入力データレコードの NAME フィールドと SCORE フィールドの間は 2 つ以上のブランクを使用して区切られている必要があります。

次に示すように、フォーマット修飾子と入力形式を併用することもできます。

```
input name $ & +3 lastname & $15. team $;
```

また、この INPUT ステートメントでは読み込むデータは同じですが、入力レコードにあるすべてのデータを読み込む必要はないことを示しています。コラムポインタコントロールが+3と指定されているので、ポインタは SCORE フィールドの位置を通過して LASTNAME フィールドと TEAM フィールドの値を読み込みます。

**例 3: 入力形式を使用した位置が揃っていないデータの読み込み**

次の DATA ステップでは、入力形式付きの修飾リスト入力にデータ値を読み込みます。

```
data jansales;
  input item : $10. amount;
  datalines;
trucks 1382
vans 1235
sedans 2391
;
```

\$10.入力形式では、文字変数の値として最大 10 文字まで読み込むことができます。

**例 4: 入力形式付きのリスト入力を使用したカンマ区切りデータの読み込み**

次の DATA ステップでは、INFILE ステートメントに DELIMITER=オプションを指定して、ブランクのかわりにカンマで区切られた値をリスト入力に読み込みます。例では入力形式を使用して日付を読み込み、出力形式を使用してその日付を書き込みます。

```
data scores2;
  length Team $ 14;
  infile datalines delimiter=';';
  input Name $ Score1-Score3 Team $ Final_Date:MMDDYY10.;
  format final_date weekdate17.;
  datalines;
Joe,11,32,76,Red Racers,2/3/2007
```

```

Mitchell,13,29,82,Blue Bunnies,4/5/2007
Susan,14,27,74,Green Gazelles,11/13/2007
;
proc print data=scores2;
  var Name Team Score1-Score3 Final_Date;
  title 'Soccer Player Scores';
run;

```

### アウトプット 2.13 カンマで区切られたデータの出力

Obs	Name	Team	Score1	Score2	Score3	Final_Date
1	Joe	Red Racers	11	32	76	Sat, Feb 3, 2007
2	Mitchell	Blue Bunnies	13	29	82	Thu, Apr 5, 2007
3	Susan	Green Gazelles	14	27	74	Tue, Nov 13, 2007

### 例 5: 修飾リスト入力を使用した区切られたデータの読み込み

次の DATA ステップでは、INFILE ステートメントに DSD オプションを指定し、INPUT ステートメントにチルダ(~)フォーマット修飾子を指定します。ここでは、文字データに含まれる引用符は保持されます。また、引用符で囲まれた文字列に含まれる区切り文字は、区切り文字ではなく文字として読み込まれます。

```

data scores;
  infile datalines dsd;
  input Name : $9. Score1-Score3
        Team ~ $25. Div $;
  datalines;
  Joseph,11,32,76,"Red Racers, Washington",AAA
  Mitchel,13,29,82,"Blue Bunnies, Richmond",AAA
  Sue Ellen,14,27,74,"Green Gazelles, Atlanta",AA
;

```

```
proc print; run;
```

実行後に作成された SCORES データセットを PROC PRINT で出力した結果を次に示します。TEAM の値には引用符が含まれています。

### アウトプット 2.14 SCORES データセット

Obs	Name	Score1	Score2	Score3	Team	Div
1	Joseph	11	32	76	"Red Racers, Washington"	AAA
2	Mitchel	13	29	82	"Blue Bunnies, Richmond"	AAA
3	Sue Ellen	14	27	74	"Green Gazelles, Atlanta"	AA

**関連項目:****ステートメント:**

- [“INFILE ステートメント” \(146 ページ\)](#)
- [“INPUT ステートメント” \(179 ページ\)](#)
- [“INPUT ステートメント、フォーマット” \(200 ページ\)](#)

---

**INPUT ステートメント、名前付き**

変数名と等号の後ろに続くデータ値を読み込み、読み込んだ値を対応する SAS 変数に割り当てます。

**該当要素:** DATA ステップ

**カテゴリ:** ファイル操作

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

**構文**

**INPUT** *<pointer-control>* *variable=* *<\$>* *<@ | @@>*;

**INPUT** *<pointer-control>* *variable= informat.* *<@ | @@>*;

**INPUT** *variable=* *<\$>* *start-column* *<-end-column>*  
*<.decimals>* *<@ | @@>*;

**引数*****pointer-control***

入力バッファ内の指定した行または列に入力ポインタを移動させます。

参照項目: [“カラムポインタコントロール” \(182 ページ\)](#) および [“行ポインタコントロール” \(184 ページ\)](#)

---

***variable=***

INPUT ステートメントで値を読み込む変数名を指定します。入力データレコードでは、次の形式でフィールドを記述します。

*variable=value*

例 [“例 3: 名前付き入力と他の入力スタイルを使用する” \(215 ページ\)](#)

---

**\$**

変数の値を数値ではなく文字値として格納するように指定します。

ヒント この変数が文字変数と事前に定義されている場合、\$を指定する必要はありません。

例 [“例 3: 名前付き入力と他の入力スタイルを使用する” \(215 ページ\)](#)

---

***informat.***

入力形式を指定します。入力形式は、入力値の読み込み方法ではなく、入力値のデータの種類を示します。



**ヒント** 変数に入力形式を関連付けるには、INFORMAT ステートメントを使用します。

**参照項目:** [SAS Viya Formats and Informats: Reference](#) の SAS 入力形式

**例** “例 3: 名前付き入力と他の入力スタイルを使用する” (215 ページ)

### **start-column**

INPUT ステートメントで変数の入力データレコードの走査を開始する列位置を指定します。変数名が始まる列位置でなくてもかまいません。

### **-end-column**

変数のデフォルトの長さを指定します。

### **.decimals**

入力値に小数点が明示的に含まれていない場合、小数点以下の桁数を指定します。

**ヒント** 入力値に小数点が明示的に含まれている場合、INPUT ステートメントで指定した小数点以下の桁数より優先されます。

### **@**

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、**後置@**と呼ばれます。

**制限事項** 後置@は、INPUT ステートメントの最後の項目として指定する必要があります。

**ヒント** 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

**参照項目:** “[ラインホールド指定子の使用](#)” (188 ページ)

### **@@**

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、**後置@@**と呼ばれます。

**制限事項** 後置@@は、INPUT ステートメントの最後の項目として指定する必要があります。

**ヒント** 後置@@は、入力行に複数のオブザベーションの値が含まれる場合に便利です。

**参照項目:** “[ラインホールド指定子の使用](#)” (188 ページ)

## **詳細**

### **名前付き入力の使用が求められる場合**

名前付き入力では、変数名と等号の後ろに変数の値が続く入力データレコードを読み込みます。INPUT ステートメントでは、入力ポインタの現在位置にある入力データレコードを読み込みます。入力データレコードの先頭に INPUT ステートメントの名前付き入力では読み込めないデータ値が含まれる場合、他の入力ス

タイトルを指定してそのデータ値を読み込むようにしてください。ただし、いったん INPUT ステートメントで名前付き入力での読み込みを開始した場合は、残りのすべての値をこの形式で読み込む必要があります。“例 3: 名前付き入力と他の入力スタイルを使用する” (215 ページ) を参照してください。

INPUT ステートメントの変数は、データレコードに値が表示されるのと同じ順序で指定する必要はありません。また、レコードに含まれる各フィールドに対して変数を指定する必要もありません。ただし、他のステートメント (ATTRIB、FORMAT、INFORMAT、LENGTH などのステートメント) で使用する変数を INPUT ステートメントに指定していなくても、入力データにその値が含まれる場合は、INPUT ステートメントによって値が自動的に読み込まれます。その場合、変数が初期化されていないことを示すメッセージがログに出力されます。

名前付き入力のすべてのデータ値に変数を指定しない場合、自動変数 \_ERROR\_ の値が 1 に設定され、メッセージがログに出力されます。次に例を示します。

```
data list;
  input name=$ age=;
  datalines;
name=John age=34 gender=M
;
```

ログに出力されるメッセージには、GENDER が定義されていないので自動変数 \_ERROR\_ の値が 1 に設定されたことが示されます。

### 制限事項

- 名前付き入力での読み込みを開始した後に、他の入力スタイルに切り替えたり、ポインタコントロールを使用することはできません。入力データレコードに含まれる残りの値はすべて、*variable=value* の形式で記述されている必要があります。名前付き入力の形式で記述されていない値は、無効なデータとして扱われます。
- 現在の入力行の後にも名前付き入力の値が続く場合、入力行の最後にスラッシュ (/) を使用します。スラッシュを指定すると、ポインタを次の行に移動させてから名前付き入力による読み込みを続けます。たとえば、この INPUT ステートメントを実行すると、次のデータレコードを読み込むことができます。

```
input name=$ age=;

name=John /
age=34
```

- ブランクが中に含まれる文字値の読み込みに名前付き入力を使用する場合、リスト入力と同じように、データ値の前後に空白を 2 つ追加します。“例 4: 値に空白が含まれる文字変数を読み込む” (215 ページ) を参照してください。
- アスタリスクや式の subscript を使用した配列参照はできません。

## 例

### 例 1: リスト入力と名前付き入力を使用する

次の DATA ステップでは、リスト入力と名前付き入力を使用して、入力データレコードを読み込みます。

```
data list;
  length name $ 20 gender $ 1;
  informat dob ddmmyy8.;
  input id name= gender= age= dob=;
```

```

datalines;
4798 name=COLIN gender=m age=23 dob=16/02/75
2653 name=MICHELE gender=f age=46 dob=17/02/73
;
proc print data=list; run;

```

この INPUT ステートメントでは、ID 変数の値をリスト入力を読み込みます。残りの変数である NAME、GENDER、AGE、DOB の値は名前付き入力を読み込みます。また、INPUT ステートメントで読み込む変数 NAME の文字値が 8 文字で切り捨てられないように、LENGTH ステートメントを指定しています。

### 例 2: 名前付き入力を変数を任意の順序で使用する

この DATA ステップでも、前述の例と同じデータを使用し、リスト入力と名前付き入力を入力データレコードを読み込みます。ただし、この例では、最初に読み込む ID の値を除き、読み込む 2 行のデータでの値の順序は異なります。

```

data list;
  length name $ 20 gender $ 1;
  informat dob ddmmyy8.;
  input id dob= name= age= gender=;
  datalines;
4798 gender=m name=COLIN age=23 dob=16/02/75
2653 name=MICHELE dob=17/02/73 age=46 gender=f
;
proc print data=list; run;

```

### 例 3: 名前付き入力と他の入力スタイルを使用する

次の DATA ステップでは、リスト入力と名前付き入力を使用して入力データレコードを読み込みます。

```

data list;
  input id name=$20. gender=$;
  informat dob ddmmyy8.;
  datalines;
4798 gender=m name=COLIN age=23 dob=16/02/75
2653 name=MICHELE age=46 gender=f
;
proc print data=list; run;

```

この INPUT ステートメントでは、最初の変数 ID の値をリスト入力を読み込みます。残りの変数である NAME、GENDER、DOB の値は名前付き入力を読み込みます。これら変数の値は、データレコードにある順序で読み込まれるわけではありません。NAME=に\$20.と入力形式を指定しているため、INPUT ステートメントでの読み込み時に文字値が 8 文字で切り捨てられることはありません。INFORMAT ステートメントで変数 DOB を参照するように指定されているため、INPUT ステートメントで DOB=フィールドの値を読み込みます。AGE=フィールドの値の読み込みはすべてスキップされます。DOB が初期化されていないこと、AGE が定義されていないこと、自動変数\_ERROR\_が 1 に設定されたことを示すメッセージが SAS ログに出力されます。

### 例 4: 値に空白が含まれる文字変数を読み込む

次の DATA ステップでは、値に空白が含まれる文字変数を名前付き入力を読み込みます。

```

data list2;
  informat header $30. name $15.;
  input header= name=;

```

```

datalines;
header= age=60 AND UP name=PHILIP
;

```

変数 HEADER の値である **AGE=60 AND UP** の前後には、2 個のスペースが挿入されています。このフィールドには、等号も含まれています。

## 関連項目:

### ステートメント:

- [“INPUT ステートメント” \(179 ページ\)](#)

---

## KEEP ステートメント

出力 SAS データセットに含める変数を指定します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
情報

**種類:** 宣言

## 構文

**KEEP** *variable-list*;

## 引数

### *variable-list*

出力データセットに書き込む変数の名前を指定します。

**ヒント** SAS で使用可能な任意の形式で変数をリストします。

## 詳細

KEEP ステートメントでは、DATA ステップで指定した変数のみを 1 つまたは複数のデータセットに書き込みます。KEEP ステートメントは、同一の DATA ステップ内で作成されたすべての SAS データセットに適用されます。また、ステップのどの位置に指定してもかまいません。KEEP または DROP ステートメントが存在しない場合、DATA ステップで作成されたすべてのデータセットにすべての変数が含まれます。

**注** 同一の DATA ステップで KEEP ステートメントと DROP ステートメントの両方を使用しないでください。

## 比較

- SAS PROC ステップでは、KEEP ステートメントは使用できません。KEEP= データセットオプションは使用できます。
- KEEP ステートメントは DATA ステートメントで指定したすべての出力データセットに適用されます。データセットごとに異なる変数を書き込むには、KEEP= データセットオプションを使用します。

- DROP ステートメントは、出力データセットから除外する変数を指定する同種のステートメントです。
- KEEP ステートメントでは出力データセットに含める変数を指定し、DROP ステートメントでは出力データセットから除外する変数を選択します。サブセット化 IF ステートメントでは、オブザベーションを選択します。
- KEEP ステートメントと RETAIN ステートメントを混同しないでください。RETAIN ステートメントは、DATA ステップの繰り返し間に変数の値を保持するものです。KEEP ステートメントは変数の値には影響を与えず、出力データセットに含める変数の指定のみ実行します。

## 例

### 例 1: KEEP ステートメントの基本的な使用法

これらの例では、KEEP ステートメントで変数をリストする場合の正しい構文を示します。

```
keep name address city state zip phone;
```

```
keep rep1-rep5;
```

### 例 2: データセットでの変数の保持

この例では、KEEP ステートメントを使用して、出力データセットに変数 NAME および AVG のみを含めます。AVG の計算に使用される変数 SCORE1 から SCORE20 は、データセット AVERAGE には書き込まれません。

```
data average;
  keep name avg;
  infile file-specification;
  input name $ score1-score20;
  avg=mean(of score1-score20);
run;
```

## 関連項目:

### データセットオプション:

- [“KEEP= Data Set Option” \(SAS Viya Data Set Options: Reference\)](#)

### ステートメント:

- [“DROP ステートメント” \(70 ページ\)](#)
- [“IF ステートメント、サブセット化” \(134 ページ\)](#)
- [“RETAIN ステートメント” \(361 ページ\)](#)

---

## LABEL ステートメント

説明を示すラベルを変数に割り当てます。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
情報

種類: 宣言

## 構文

**LABEL** *variable-1=label-1...<variable-n=label-n>;***LABEL** *variable-1=' ' ...<variable-n=' '>;*

## 引数

### *variable*

ラベルを付ける変数を指定します。

ヒント ラベルと変数の組み合わせを複数指定することができます。

### *label*

ラベルの文字列をブランクも含めて 256 文字以内で指定します。

制限事項 ラベルにセミコロン(;)や等号(=)が含まれている場合は、ラベルのテキストを一重引用符または二重引用符で囲む必要があります。

ラベルに一重引用符(')が含まれている場合は、ラベルのテキストを二重引用符で囲む必要があります。

ActiveX と Java のデバイスでは、変数名の置き換えを一部サポートしています。変数に指定したラベルのテキストが許可されたスペースを超える場合、軸や凡例のタイトルをラベリングするプロシジャのかわりに、変数名が使用されます。SAS/GRAPH GPLOT プロシジャを除き、ActiveX または Java デバイスが指定されていない場合、変数名は使用されません。

注 LABEL ステートメントは、最初の変数の後に等号(=)があるものと想定します。それ以外の文字が見つかった場合、エラーが発生します。LABEL ステートメントは、最初の変数の直後に続く等号の後から、等号が直後に続く別の変数に達するまでの区間に存在するすべての文字を、引用符の有無にかかわらず、ラベルの一部として見なします。次の例では、変数 *x* のラベルは、**this is x y 4 =this is y** になります。なぜなら、等号が直後に続く変数は *z* であるためです。

```
data test1;
  x=1;
  y=1;
  z=1;
  label x="this is x" y 4 ="this is y" z="This is z";
run;
```

次のような LABEL ステートメントでも、変数 *x* のラベルは上記の例と同じになります。

```
label x=this is x y 4 = this is y z=This is z;
```

ヒント ラベルと変数の組み合わせを複数指定することができます。

,,

変数からラベルを削除します。ブランク 1 つを引用符で囲むと、指定済みのラベルが取り消されます。

## 詳細

DATA ステップ内で LABEL ステートメントを使用すると、変数を含む SAS データセットのディスクリプタ情報が変更され、ラベルとその変数は恒久的に関連付けられます。1 つの LABEL ステートメントで複数の変数にラベルを割り当てることができます。

PROC ステップでも LABEL ステートメントを使用できますが、ルールは異なります。詳細については、[SAS Viya Data Management and Utility Procedures Guide](#) を参照してください。

## 比較

ATTRIB ステートメントと LABEL ステートメントは、どちらも変数へのラベルの割り当てや、変数が割り当てられたラベルの変更を実行することができます。

## 例

### 例 1: ラベルの指定

いくつかの LABEL ステートメントの例を次に示します。

- label compound=Type of Drug;
- label date="Today's Date";
- label n='Mark's Experiment Number';
- label score1="Grade on April 1 Test"  
score2="Grade on May 1 Test";

### 例 2: ラベルの取り消し

定義済みのラベルを取り消す例を次に示します。

```
data rtest;
  set rtest;
  label x='';
run;
```

## 関連項目:

### ステートメント:

- [“ATTRIB ステートメント” \(29 ページ\)](#)

---

## ラベル: ステートメント

他のステートメントから参照されるステートメントを識別します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

種類: 宣言

## 構文

*label*: *statement*;

## 引数

### *label*

SAS 名を指定します。SAS 名の後ろにはコロン(:)が必要です。 *label* 引数は必ず指定する必要があります。

### *statement*

ヌルステートメント(;)など、実行ステートメントを指定します。 *statement* 引数は必ず指定する必要があります。

**制限事項** DATA ステップ内では、同一のラベルを複数のステートメントに指定できません。

DATA ステップ内のステートメントにラベルを設定する場合、同じステップ内のステートメントやオプションからはそのラベルを参照する必要があります。

**ヒント** ヌルステートメントにもラベルを設定できます。たとえば、  
ABC;  
のように指定します。

## 詳細

ステートメントラベルは、GO TO ステートメント、LINK ステートメント、FILE ステートメントの HEADER=オプション、INFILE ステートメントの EOF=オプションのいずれかで参照されるステートメントを識別します。

## 比較

LABEL ステートメントは、記述的なラベルを変数に割り当てます。ステートメントラベルは、同一の DATA ステップ内で GO TO ステートメントなどの他のステートメントから参照される 1 つのステートメントまたはステートメントのグループを識別します。

## 例: 別のステートメントへの移動

この例では、Stock=0 であれば、GO TO ステートメントは SAS を reorder というラベルが設定されているステートメントに移動します。Stock の値が 0 でなければ、RETURN ステートメントまで処理を続行した後、次のオブザベーションを処理するために DATA ステップの先頭に戻ります。

```
data Inventory Order;
  input Item $ Stock @;
  /* go to label reorder: */
  if Stock=0 then go to reorder;
  output Inventory;
  return;
  /* destination of GO TO statement */
  reorder: input Supplier $;
```



```

put 'ORDER ITEM ' Item 'FROM ' Supplier;
output Order;
datalines;
milk 0 A
bread 3 B
;

```

## 関連項目:

### ステートメント:

- [“GO TO ステートメント” \(132 ページ\)](#)
- [“LINK ステートメント” \(266 ページ\)](#)

### ステートメントオプション:

- [“EOF=label” \(152 ページ\)](#)(INFIL ステートメント)
- [“HEADER=label” \(84 ページ\)](#)オプション(FILE ステートメント)

---

## LEAVE ステートメント

現在のループの処理を中止し、シーケンス内の次のステートメントから再開します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 実行

---

## 構文

**LEAVE;**

### 引数なし

LEAVE ステートメントは現在の DO ループや SELECT グループの処理を中止し、この DO ループや SELECT グループの後に続く次のステートメントから DATA ステップの処理を再開します。

### 詳細

LEAVE ステートメントを使用すると、条件に基づいて DO ループまたは SELECT グループを途中で終了することができます。

### 比較

- LEAVE ステートメントは、現在のループの処理を終了します。CONTINUE ステートメントは、ループの現在の繰り返し処理を中止し、次の繰り返し処理から再開します。
- DO ループまたは SELECT グループ内で LEAVE ステートメントを使用します。CONTINUE ステートメントは DO ループのみで使用できます。

## 例: 指定した条件で DO ループの処理を中止する

次の DATA ステップでは、LEAVE ステートメントを使用して DO ループの処理を中止します。この例では、IF/THEN ステートメントを使用して BONUS の値をチェックします。BONUS の値が指定した最大値である 500 になると、LEAVE ステートメントは DO ループの処理を中止します。

```
data week;
  input name $ idno start_yr status $ dept $;
  bonus=0;
  do year= start_yr to 1991;
    if bonus ge 500 then leave;
    bonus+50;
  end;
  datalines;
Jones 9011 1990 PT PUB
Thomas 876 1976 PT HR
Barnes 7899 1991 FT TECH
Harrell 1250 1975 FT HR
Richards 1002 1990 FT DEV
Kelly 85 1981 PT PUB
Stone 091 1990 PT MAIT
;
```

### 関連項目:

#### ステートメント:

- [“DO ステートメント” \(62 ページ\)](#)
- [“SELECT ステートメント” \(375 ページ\)](#)

---

## LENGTH ステートメント

変数の保存時に使用するバイト数を指定します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
情報

**種類:** 宣言

**注意:** **小数を含んでいる数値の短縮化を防ぎます。** 変数に小数が含まれる場合、数値変数の有効桁数はその数値変数の長さに密接に関連します。整数を含む変数の長さは、各動作環境向けの SAS ドキュメント内のルールに従って短くすることができます。ただし、小数を含む変数の長さを短くすると、重要な有効桁数が削除される場合があります。

---

### 構文

**LENGTH** *variable-specification(s)* <DEFAULT=*n*>;

## 引数

### *variable-specification*

この引数は必須です。次の形式を使用します。

*variable(s)*<\$>*length* | <VARCHAR(*length* | \*)>

### *variable*

長さを割り当てる 1 つまたは複数の変数(DATA ステップ内の任意の変数を含む)を指定します。出力データセットからは除外される変数も含まれます。

**制限事項** 配列参照は指定できません。

**ヒント** 指定した変数が文字変数の場合、長さはプログラムデータベクトルと出力データセットに適用されます。指定した変数が数値変数の場合、長さは出力データセットにのみ適用されます。

### \$

直前にある変数が、CHAR データタイプの文字変数であることを指定します。

**デフォルト** 変数は数値変数として処理されます。

### VARCHAR

直前にある変数が、VARCHAR データタイプの文字変数であることを指定します。

### *length*

数値定数を指定します。この値は、変数の値を保存するときに使用するバイト数を示します。

**範囲** CHAR 変数の場合、1 から 32767 まで VARCHAR 変数の場合、1 から  $1 - ((2^{31} - 1) / 4)$  まで数値変数の場合、3 から 8 まで数値変数に指定できる長さの最小値は、ユーザーのシステムが使用する浮動小数点形式によって異なります。ほとんどのシステムでは IEEE 浮動小数点形式が使用され、最小値は 3 バイトです。

**制限事項** VARCHAR 変数に文字定数を割り当てる場合、その文字定数は 32767 バイトまでに制限されます。

### \*

許可されている最大長(現時点で  $2^{31} - 1$ )を使用することを意味します。

### DEFAULT=*n*

新しく作成する数値変数の値を保存するときに使用するデフォルトのバイト数を変更します。

**デフォルト** 8

**範囲** 3 から 8

## 詳細

### 一般的な情報

通常、変数の長さは次の情報によって決定されます。

- 文字変数または数値変数

- 変数の作成方法
- LENGTH ステートメントまたは ATTRIB ステートメントの有無

長さを割り当てるルール(VARCHAR 変数に関する例外を含む)に基づいて、LENGTH ステートメントで割り当てた変数の長さを ATTRIB ステートメントで変更できます。またその逆も実行できます。変数への長さの割り当てに関しては、次のトピックを参照してください。

- [“SAS Cloud Analytic Services Data Types” \(SAS Cloud Analytic Services: Language Reference\)](#)
- [“Create a VARCHAR Variable Using the LENGTH Statement” \(SAS Cloud Analytic Services: Accessing and Manipulating Data\)](#)
- [“VARCHAR Data Type in String Functions” \(SAS Viya National Language Support: Reference Guide\)](#)

## 比較

ATTRIB ステートメントを使用すると、変数の属性と同様に変数の長さも割り当てることができます。

## 例

この例では、LENGTH ステートメントを使用して文字変数 NAME の長さを 25 バイトに設定します。また、この LENGTH ステートメントは、新しく作成する数値変数を保存するときに使用するデフォルトのバイト数を 8 バイトから 4 バイトに変更します。TRIM 関数は、LASTNAME の末尾にある空白を削除してから、結合します。

- カンマ(,)
- ブランク
- FIRSTNAME の値

LENGTH ステートメントを省略すると、変数 NAME の長さは 32 バイトに設定されます。

```
data testlength;
  informat FirstName LastName $15. n1 6.2;
  input firstname lastname n1 n2;
  length name $25 default=4;
  name=trim(lastname)||','||firstname;
  datalines;
Alexander Robinson 35 11
;
proc contents data=testlength;
run;
proc print data=testlength;
run;
```

次の出力結果は、PROC CONTENTS の実行結果の一部と PROC PRINT で生成したレポートを示しています。

アウトプット 2.15 TESTLENGTH に対する PROC CONTENTS の実行結果の一部

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Informat
1	FirstName	Char	15	\$15.
2	LastName	Char	15	\$15.
3	n1	Num	4	6.2
4	n2	Num	4	
5	name	Char	25	

アウトプット 2.16 変数の長さの設定

Obs	FirstName	LastName	n1	n2	name
1	Alexander	Robinson	0.35000	11	Robinson, Alexander

**関連項目:**

- PROC ステップで LENGTH ステートメントを使用する方法の詳細については、[SAS Viya Data Management and Utility Procedures Guide](#) を参照してください。
- [“Data Representation” \(Batch and Line Mode Processing in SAS Viya\)](#)

**ステートメント:**

- [“ATTRIB ステートメント” \(29 ページ\)](#)

## LIBNAME ステートメント

SAS ライブラリへのライブラリ参照名(ショートカット名)の関連付けや関連付けの取り消し、1 つまたはすべてのライブラリ参照名の関連付けの取り消しを実行します。また、SAS ライブラリの属性のリスト、SAS ライブラリの連結や SAS カタログの連結を実行します。

**該当要素:** 任意の場所

**カテゴリ:** データアクセス

**制限事項:** このステートメントは CAS Server ではサポートされていません。

SAS が Linux でロックダウン状態にある場合、ロックダウンパスリストに含まれていないファイルに関しては、LIBNAME ステートメントを使用できません。

### 構文

形式 1: **LIBNAME** *libref* <engine> 'SAS-library' <options> <engine/host-options>;

形式 2: **LIBNAME** *libref* CLEAR | \_ALL\_ CLEAR ;

形式 3: **LIBNAME** *libref* LIST | *\_ALL\_* LIST;

形式 4: **LIBNAME** *libref* <*engine*> (*library-specification-1* <...*library-specification-n*>) <*options*>;

## 引数

### *libref*

SAS ファイルを格納する集約記憶域のショートカット名またはニックネームを指定します。新しいライブ参照名を割り当てるときは、任意の SAS 名を指定します。SAS ライブラリからファイル参照名の関連付けを取り消したり、属性を出力する場合は、割り当て済みのライブラリ参照名を指定します。

範囲 1~8 バイト

ヒント ライブラリ参照名と SAS ライブラリの関連付けは、SAS セッション終了まで維持されるか、または他の LIBNAME ステートメントで関連付けの変更や関連付けの取り消しを実行するまで維持されます。

### 'SAS-library'

SAS ライブラリの物理名を指定する必要があります。物理名には動作環境で判別できる名前を指定します。物理名は一重引用符または二重引用符で囲みます。

### *library-specification*

2 つ以上の SAS ライブラリを指定します。物理名で指定したライブラリ、すでにライブラリ参照名を割り当て済みのライブラリ、または、1 つのライブラリ参照名でアクセスしようとする名前またはライブラリ参照名を組み合わせたライブラリが対象になります。ライブラリを連結したいときは、LIBNAME ステートメントのこれらの形を使用します。各ライブラリをブランクまたはカンマで区切ってから、リスト全体を丸かっこで囲みます。

### 'SAS-library'

SAS ライブラリの物理名を引用符で囲んで指定します。

SAS-library は指定するエンジンおよび現在使用している作業ディレクトリに依存します。表 2.7 (237 ページ) に、各エンジンがこの引数に何を指定するものと想定しているか、を説明します。Linux ディレクトリのパス名を指定してください。LIBNAME ステートメントを使用してディレクトリを作成することはできません。指定するディレクトリはすでに存在している必要があります、ユーザーがそのディレクトリへのアクセス権を持っている必要があります。Linux のパス名は大文字と小文字を区別します。

### *libref*

割り当て済みのライブラリ参照名の名前を指定します。ライブラリ参照名は引用符で囲まないでください。

制限事項 ライブラリを連結するときは、エンジンまたは動作環境に固有のオプションは指定できません。

参照項目: “[ライブラリ連結のルール](#)” (240 ページ)

例 “[例 2: SAS ライブラリの論理的連結](#)” (241 ページ)

### *engine*

エンジンの名前を指定します。

ヒント 通常、SAS ではライブラリ内のファイルへのアクセスに使用する適切なエンジンを自動的に検出します。デフォルトエンジン以外のエンジ

ンを使用して新しいライブラリを作成する場合、この自動選択を無効にできます。

### options

LIBNAME ステートメントのオプションです。このオプションの詳細については、「[LIBNAME オプション](#)」(227 ページ)を参照してください。

### engine/host-options

「[Linux オプション](#)」(234 ページ)に説明しているどのオプションでも構いません。

### CLEAR

指定されたライブラリ参照名をクリアします。`_ALL_`を指定した場合は、現在定義されているライブラリ参照名をすべてクリアします。Sashelp と Work は割り当てられたままです。

注: Linux の環境変数によって定義されているライブラリ参照名をクリアすると、変数は定義されたままになりますが、ライブラリ参照名としては認識されません。その変数を、ライブラリ参照名またはファイル参照名としてくりかえし使用できます。

ライブラリ参照名とそれに対応するライブラリとの関連付けは、ジョブまたはセッションの終了時に自動的にクリアされます。現在のセッション中に既存のライブラリ参照名を別の SAS ライブラリと関連付ける場合、セッションを終了したりライブラリ参照名をクリアしたりする必要はありません。新規の SAS ライブラリの LIBNAME ステートメントを発行するときに、ライブラリ参照名が自動的に再割り当てされます。

### `_ALL_`

現在割り当てられているすべてのライブラリ参照名に対して、CLEAR 引数または LIST 引数を適用するように指定します。

### LIST

1 つまたは複数の SAS ライブラリの属性を SAS ログに書き込みます。

ヒント 1 つの SAS ライブラリの属性をリストするには、`libref` を指定します。  
ト 現在のセッションにあるライブラリ参照名を含むすべての SAS ライブラリの属性をリストするには、`_ALL_` を指定します。

## LIBNAME オプション

### ACCESS=READONLY|TEMP

#### READONLY

SAS ライブラリ全体に読み取り専用属性を割り当てます。このライブラリ内のデータセットを開いて、情報の更新や新しい情報の書き込みを行うことができなくなります。

#### TEMP

SAS ライブラリを一時作業ライブラリとして扱う場合に指定します。システムは、CPU サイクルを消費しないことにより、TEMP ライブラリにあるファイルが破損されないようにします。

ヒント データが修復可能な場合にのみ、リソースを節約するために ACCESS=TEMP を使用します。

### COMPRESS=NO | YES | CHAR | BINARY

SAS ライブラリの出力 SAS データセットにあるオブザベーションの圧縮を制御します。

**NO**

新しく作成する SAS データセット内のオブザベーションを圧縮しないように指定します(固定長レコード)。

**YES | CHAR**

RLE (Run Length Encoding)を使用して、新しく作成する SAS データセット内のオブザベーションを圧縮するように指定します(可変長レコード)。RLE では、連続する文字の繰り返し(ブランクを含む)を 2 バイトまたは 3 バイト形式に減らすことによって、オブザベーションを圧縮します。

ヒント この圧縮アルゴリズムは文字データに対して使用します。

**BINARY**

RDC(Ross Data Compression)を使用して、新しく作成する SAS データセット内のオブザベーションを圧縮するように指定します(可変長レコード)。RDC では、ランレングスエンコーディングとスライディングウィンドウ圧縮を結合して、ファイルが圧縮されます。

ヒ  
ン  
ト この方式は、中サイズから大サイズ(数百バイトまたはそれ以上)のバイナリデータ(数値変数)のブロックを圧縮する場合に有効です。圧縮機能では一度に 1 レコードのみが処理されるので、効率的な圧縮には、レコード長が数百バイトまたはそれ以上である必要があります。

**CVPBYTES=bytes**

トランスコーディングを必要とする SAS データファイルを処理するときに、拡張する文字変数の長さをバイト数で指定します。CVP エンジンは、長さを拡張することにより、文字データの切り捨てが発生しないようにします。文字変数の長さは、指定した値を現在の長さに加えて増やします。0 から 32766 までの値を指定できます。

たとえば、次の LIBNAME ステートメントでは、CVPBYTES=オプションを指定して CVP エンジンを暗黙的に割り当てています。

```
libname expand 'SAS data-library' cvpbytes=5;
```

文字変数の長さは、5 バイト追加して増やします。長さが 10 の文字変数は長さ 15 に、長さが 100 の文字変数は長さ 105 に増加します。

デ  
フォ  
ルト CVPBYTES=を指定すると、指定値に従って文字変数の長さを拡張するために CVP エンジンを自動的に使用します。CVP エンジンを明示的に割り当てても、CVPBYTES=または CVPMULTIPLIER=を指定しない場合は、CVPMULTIPLIER=1.5 を使用して文字変数を長くします。

制限  
事項 CVP エンジンは、SAS データファイルをサポートしていますが、SAS のビュー、カタログおよびアイテムストアなどはサポートしていません。

CVP エンジンは、入力(読み込み)処理でのみ使用できます。

CVP エンジンなどのエンジンが混在するライブラリ連結では、SAS データファイルのみが処理されます。たとえば、COPY プロシジャを実行する場合は、SAS データファイルのみコピーされます。

要件 指定するバイト数は、どのような拡張にも適応できる大きさにする必要があります。この大きさが不十分だと、データが切り捨てられ、その結果 SAS ログにエラーメッセージが書き込まれます。



**操作** CVPBYTES=と CVPMULTIPLIER=の両方を指定できません。いずれか1つのオプションを指定してください。

**参照項目:** [“Avoiding Character Data Truncation By Using the CVP Engine” \(SAS Viya National Language Support: Reference Guide\)](#)

### **CVPENGINE=engine**

トランスコーディングを必要とする SAS データファイルを処理するために使用するエンジンを指定します。トランスコードする文字変数の長さが CVP エンジンによって拡張されるため、文字データの切り捨ては発生しません。次に、指定されたエンジンで実際のファイル処理を実行します。

**別名** CVPENG

**デフォルト** SAS は、デフォルトの SAS エンジンを使用します。

**参照項目:** [“Avoiding Character Data Truncation By Using the CVP Engine” \(SAS Viya National Language Support: Reference Guide\)](#)

### **CVPMULTIPLIER=multiplier**

トランスコーディングを必要とする SAS データファイルを処理するときに、文字変数の長さを拡張するための乗数値を指定します。CVP エンジンは、長さを拡張することにより、文字データの切り捨てが発生しないようにします。文字変数の長さは、指定した値を現在の長さに乗算して増やします。1 から 5 までの乗数値を指定できます。

たとえば、次の LIBNAME ステートメントでは、CVPMULTIPLIER=オプションを指定して CVP エンジンを暗黙的に割り当てています。

```
libname expand 'SAS data-library' cvpmultiplier=2.5;
```

文字変数の長さは 2.5 倍になります。長さが 10 の文字変数は長さ 25 に、長さが 100 の文字変数は長さ 250 に増加します。

**別名** CVPMULT

**デフォルト** CVPMULTIPLIER=を指定すると、指定値に従って文字変数の長さを拡張するために CVP エンジンを自動的に使用します。CVP エンジンを明示的に割り当てても、CVPMULTIPLIER=または CVPBYTES=を指定しない場合は、CVPMULTIPLIER=1.5 を使用して文字変数を長くします。

**制限事項** CVP エンジンは、SAS データファイルをサポートしていますが、SAS のビュー、カタログおよびアイテムストアなどはサポートしていません。

CVP エンジンは、入力(読み込み)処理でのみ使用できます。

CVP エンジンなどのエンジンが混在するライブラリ連結では、SAS データファイルのみが処理されます。たとえば、COPY プロシジャを実行する場合は、SAS データファイルのみコピーされます。

**要件** 指定するバイト数は、どのような拡張にも適応できる大きさにする必要があります。この大きさが不十分だと、データが切り捨てられ、その結果 SAS ログにエラーが書き込まれます。

**操作** CVPMULTIPLIER=と CVPBYTES=の両方を指定できません。いずれか1つのオプションを指定してください。

参照 [“Avoiding Character Data Truncation By Using the CVP Engine”](#)  
 項目: [\(SAS Viya National Language Support: Reference Guide\)](#)

**CVPVARCHAR=YES | NO**

入力ファイル処理の際に、固定幅文字変数を可変幅文字に変換するか否かを指定します。固定幅文字変数のバイト長は、SAS セッションエンコーディングにおける 1 文字当たり最大バイト数に、指定された固定幅文字長を掛けた値です。

デフォルト No

操作 CVPVARCHAR=YES を指定した場合、CVPMULTIPLIER=オプションや CVPBYTES=オプションを指定しても無視されます。

注 CHAR カラム以下にある文字列データの末尾に空白があれば削除します。

TRANSCODE=NO という形式の固定幅文字変数は、変換の対象外です。

**EXTENDOBSCOUNTER=YES | NO**

SAS ライブラリの出力 SAS データファイル内の最大オブザベーション数を拡張するかどうかを指定します。

**YES**

新しく作成された SAS データファイルに、32 ビット長の制限を超えてオブザベーションをカウントする拡張ファイル形式をリクエストします。32 ビット長整数を使用してオブザベーション数を格納する動作環境用に作成された SAS データファイルであっても、カウンタに関しては 64 ビットファイルと同じように動作します。これがデフォルトの設定です。

制限事項 拡張オブザベーションカウントで作成した SAS データファイルは、SAS9.3 より前のリリースとは互換性がありません。

BASE Engine でのみ使用してください。

拡張したオブザベーション数の属性は新しいファイルに継承することはできません。新規のファイルを拡張オブザベーションカウント属性なしで作成した場合、そのファイルには EXTENDOBSCOUNTER=NO を指定してください。

**NO**

動作環境の長整数のサイズから、新しく作成された SAS データファイル内のオブザベーションの最大数を設定するように指定します。32 ビット長整数の動作環境では、オブザベーションの最大数は  $2^{31}-1$ 、または約 20 億(2,147,483,647)になります。64 ビット長整数の動作環境では、オブザベーションの最大数は  $2^{63}-1$ 、または約 920 京になります。

別名 EOC=

デフォルト YES

**INENCODING=ANY | ASCIIANY | EBCDICANY | *encoding-value***

SAS ライブラリ内の SAS データセットの読み込み(入力処理)時にエンコーディングを無効にします。

参照項目: [“INENCODING= and OUTENCODING= Options” \(SAS Viya National Language Support: Reference Guide\)](#)

### OUTENCODING=

OUTENCODING=ANY | ASCIIANY | EBCDICANY | *encoding-value*

SAS ライブラリ内で SAS データセットの作成(出力処理)時にエンコーディングを無効にします。

参照項目: [“INENCODING= and OUTENCODING= Options” \(SAS Viya National Language Support: Reference Guide\)](#)

### OUTREP=*format*

SAS ライブラリのデータ形式を指定します。このデータ形式には、特定の動作環境でデータを格納する形式を指定します。使用する標準や方式は、動作環境によって異なります。たとえば、浮動小数点数の標準や格納方式(IEEE か IBM メインフレーム方式か)、文字エンコーディング(ASCII か EBCDIC か)、メモリ上のバイトオーダー(ビッグエンディアンかリトルエンディアンか)、ワードラインメント(4 バイト境界か 8 バイト境界か)、整数データ型の長さ(16 ビット、32 ビット、64 ビットのいずれか)、倍精度変換方式(バイトスワップを行うかどうか)などは、動作環境ごとに決定されます。

デフォルトでは、SAS を実行している CPU のデータ形式を使用して新しいデータセットが作成されます。OUTREP=オプションを指定すると、別のデータ形式の SAS データセットを作成できます。たとえば、Linux 環境で Windows のデータ形式の SAS データセットを作成できます。

次の表に OUTREP=の値を示します。

表 2.6 OUTREP=オプションでのデータ表現の値

OUTREP=の値	エイリアス*	環境
ALPHA_TRU64	ALPHA_OSF	Tru64 UNIX
ALPHA_VMS_32	ALPHA_VMS	OpenVMS Alpha
ALPHA_VMS_64		OpenVMS Alpha
HP_IA64	HP_ITANIUM	Itanium プロセッサファミリーアーキテクチャ向け HP-UX
HP_UX_32	HP_UX	PA-RISC 向け HP-UX
HP_UX_64		PA-RISC 向け HP-UX(64 ビット版)
INTEL_ABI		Intel アーキテクチャ向け ABI
LINUX_32	LINUX	Intel アーキテクチャ向け Linux
LINUX_IA64		Itanium ベースシステム向け Linux
LINUX_X86_64		x64 向け Linux
MIPS_ABI		MIPS ABI

OUTREP=の値	エイリアス*	環境
MVS_32	MVS	z/OS 上の 31 ビット版 SAS
MVS_64_BFP		z/OS 上の 64 ビット版 SAS
OS2		Intel 向け OS/2
RS_6000_AIX_32	RS_6000_AIX	AIX
RS_6000_AIX_64		AIX
SOLARIS_32	SOLARIS	SPARC 向け Solaris
SOLARIS_64		SPARC 向け Solaris
SOLARIS_X86_64		x64 向け Solaris
VAX_VMS		OpenVMS VAX
VMS_IA64		HP Integrity 上の OpenVMS
WINDOWS_32	WINDOWS	Microsoft Windows 上の 32 ビット版 SAS
WINDOWS_64		Microsoft Windows 上の 32 ビット版 SAS (Itanium ベースシステムおよび x64 システム向け)

\* 現在の値を使用することをお勧めします。エイリアスは互換性のためにだけ使用します。

**操作** エンコーディングに互換性がない場合にトランスコーディングを行うと、文字データが失われることがあります。詳細については、*SAS Viya 各国語サポート: リファレンスガイド*を参照してください。

### POINTOBS=YES | NO

圧縮されたデータセットを作成するかどうかを指定します。このデータセットのオブザベーションはランダムにも順次にもアクセスできます。

#### YES

オブザベーション番号でランダムにアクセスされる圧縮データセットを作成します。

**注** 各データセットでは、POINTOBS=データセットオプションは、LIBNAME ステートメントに指定した REPEMPTY=オプションの設定より優先されます。

**ヒント** POINTOBS=YES を指定しても、データセットからデータを取得する効率に影響はありません。圧縮データセットを作成してそれを更新したりや情報を追加したりする場合、CPU の使用率が 10% 増えます。

**NO**

圧縮されたデータセットのオブザベーションをオブザベーション番号でランダムにアクセスする機能を抑制します。

ヒ POINTOBS=NO は圧縮データセット内のオブザベーションに番号  
ン で直接参照することが重要でないアプリケーションで指定すると最  
ト 適です。オブザベーション番号でデータにアクセスする必要がない  
場合は、圧縮データセットを作成またはオブザベーションを追加す  
る時に POINTOBS=NO を指定するとパフォーマンスが約 10%向上  
します。

デフォルト YES

**REPEMPTY=YES|NO**

新しい出力データが空の場合に、同じ名前の一時 SAS データセットや永久 SAS データセットの置き換えを制御します。

**YES**

指定した名前を持つ空の新しいデータセットを同じ名前を持つ既存のデータセットに置き換えます。これがデフォルトの設定です。

操作 REPEMPTY=YES に設定され、REPLACE=NO に設定されている場合、データセットは置き換えられません。

**NO**

指定した名前を持つ空の新しいデータセットは同じ名前を持つ既存のデータセットに置き換えられません。

ヒ REPEMPTY=NO を使用すると、既存のデータセット MYLIB.B を誤っ  
ン て作成された空の新しいデータセット MYLIB.B に置き換える次のよ  
ト うな構文エラーを回避できます。

```
libname libref SAS-library REPEMPTY=NO;
data mylib.a set mylib.b;
```

既存のデータセットをデータを含む新しいデータセットに置き換え、既存のデータセットを誤って作成した空の新しいデータセットで上書きしないように保護するには、REPLACE=YES と REPEMPTY=NO を設定します。

注 各データセットでは、REPEMPTY=データセットオプションは、LIBNAME ステートメントに指定した REPEMPTY=オプションの設定より優先されます。

参照項目: [“REPEMPTY= Data Set Option” \(SAS Viya Data Set Options: Reference\)](#)

**エンジン/ホストオプション****engine-host-options**

*keyword=value* という一般的な形式でリストされる 1 つ以上のオプションです。

制限事項 ライブラリを連結するときは、エンジンまたは動作環境に固有のオプションは指定できません。

## Linux オプション

### ENABLEDIRECTIO

LIBNAME ステートメントで特定されたライブラリ内で開かれるすべてのファイルに対して、ダイレクト I/O を使用可能にする指定です。

**操作** ライブラリ参照名が LIBNAME ステートメントにリストされているファイル(複数ある場合も含む)のダイレクト I/O を有効化するには、ENABLEDIRECTIO オプションと USEDIRECTIO=オプションを併用する必要があります。次の例では、ENABLEDIRECTIO オプションと USEDIRECTIO= LIBNAME オプションが併用されています。この場合、ライブラリ参照名のテストで参照されているファイルはすべて、ダイレクト I/O が可能な状態になっています。

```
LIBNAME test '! ' ENABLEDIRECTIO USEDIRECTIO=yes;
```

**注** ENABLEDIRECTIO を Work ディレクトリに対して指定することはできません。

ENABLEDIRECTIO オプションを指定してディレクトリに割り当てたライブラリ参照名が、このオプションなしで同じディレクトリに割り当てた別のライブラリ参照名と一致していません。2つのライブラリ参照名が同じディレクトリを指すことはありえますが、一方のライブラリ参照名で開いたファイルを、ダイレクト I/O で読み書きすることはできません。他のライブラリ参照名を使って開いたファイルは、通常のディスク I/O コールを使って読み書きしてください。

**ヒント** 次の例では、ENABLEDIRECTIO LIBNAME オプションを使用し、ライブラリ参照名 test に関連付けられたファイルをダイレクト I/O のために開けるようにします。USEDIRECTIO=データセットオプションは、ダイレクト I/O で test.file1 を開きます。test.file2 はダイレクト I/O が可能であっても、ダイレクト I/O では開かれません。

```
LIBNAME test ' ' ENABLEDIRECTIO;
data test.file1(USEDIRECTIO=yes);
... more SAS statements ...
run;
data test.file2;
... more SAS statements ...
run;
```

### FILELOCKS=NONE | FAIL | CONTINUE

LIBNAME ステートメントのライブラリ参照名の下で開いているファイルについて、ファイルのロックを有効化するか無効化するかを指定します。FILELOCKS ステートメントオプションは、ライブラリ参照名に関連付けられたファイルに対してのみ適用されるという点を除けば、FILELOCKS システムオプションと同様に動作します。FILELOCKS ステートメントオプションでは次の値が使用できます。

#### NONE

ファイルのロックをオフにします。NONE は、SAS がファイルでの既存のロックを確認せずにファイルを開く指定です。NONE では、ファイルにオペレーティングシステムロックがかかりません。このようなファイルは、共有更新アクセスから保護されません。

#### FAIL

ファイルのロックをオンにします。FAIL は、SAS がファイルにオペレーティングシステムロックをかけようとする指定です。ファイルへのアク



セスは、ファイルがすでにロックされている場合、またはファイルをロックできない場合に拒否されます。

### CONTINUE

ファイルのロックをオンにします。CONTINUE は、SAS がファイルにオペレーティングシステムロックをかけようとする指定です。ファイルがすでに他のユーザーにロックされている場合は、ファイルを開くことができません。何か他の理由でファイルをロックできない場合でも、ファイルを開くことは可能ですが、警告メッセージがログに出力されます。例えばファイルシステムがロックに未対応であることが考えられます。

LIBNAME ステートメントの FILELOCKS オプションは、LIBNAME ステートメントに指定したライブラリ参照名を使って開けば、データセットやカタログなどほとんどの SAS I/O ファイルに適用できます。

FILELOCKS システムオプションを使用するばあい、RESET は有効な値になりますが、FILELOCKS ステートメントオプションの場合、RESET は有効な値にはなりません。

FILELOCKS ステートメントオプションのかわりに FILELOCKS システムオプションを使用して、ユーザーのファイルにロック動作を設定します。(FILELOCKS ステートメントオプションは、SAS の今後のリリースでは重要度が低くなります。)LIBNAME ステートメントの FILELOCK オプションは LIBNAME システムオプションをオーバーライドします。詳細については、FILELOCKS システムオプションを参照してください。Scalable Performance Data Server でサポートされているオプションを指定することもできます。このオプションについては「*Scalable Performance Data Server: ユーザーズガイド*」を参照してください。

### FILELOCKWAIT=*n*

ロックされたファイルが別のプロセスで使用可能になるまで SAS が待機する秒数を設定します。

ロックされたファイルが *n* で指定した秒数が経過する前にロック解除された場合、そのファイルは現在のプロセスのためにロックされ、処理が続行します。指定した秒数に達してもファイルがロックされたままの場合、SAS はロックされたファイルのエラーをログに書き込み、DATA ステップは失敗します。

デフォルト 0

範囲 0-600

### NOSETPERM

ライブラリメンバが同一のライブラリ参照名で開かれるとき、あるライブラリメンバから別のライブラリメンバへ権限の設定が引き継がれないように指定します。パスに対して 2 つの割り当て(NOSETPERM オプションのあるものとないもの)がある場合、それらの割り当てはパスが一致しないものとして処理されます。NOSETPERM オプションがある LIBNAME ステートメントでは、権限の設定は引き継がれません。

NOSETPERM オプションを使用してライブラリ参照名の権限の設定を無効化すると、そのオプションは、ユーザーがライブラリ参照名を使用するとき常に有効になります。NOSETPERM オプションを無効化するオプションはありません。NOSETPERM オプションを無効化するには、次のステートメントを送信してください。

```
libname libref clear;
```

**TRANSFERSIZE=*n*K | *n*M**

開いているファイルから読み込まれるデータの大きなブロックのサイズを指定します。

*n*  
整数を指定します。

**K**  
ブロックのサイズをキロバイト単位で指定します。

**M**  
ブロックのサイズをメガバイト単位で指定します。

**操作** TRANSFERSIZE オプションを使用するには、ファイルをダイレクト I/O 用に開く必要があります。つまり、ENABLEDIRECTIO オプションと USEDIRECTIO=オプションを両方とも有効にする必要があります。ENABLEDIRECTIO オプションおよび USEDIRECTIO=オプションなしで TRANSFERSIZE オプションを使うと、そのオプションは受け付けられませんが機能しません。

**例** 次の例では、test.file1 がダイレクト I/O 用に開かれるため、128k ブロックのデータがこのファイルから読み込まれます。est.file2 はダイレクト I/O 用に開かれず、TRANSFERSIZE オプションはこのファイルには影響しません。

```
LIBNAME test'. 'ENABLEDIRECTIO TRANSFERSIZE=128k;
data test.file1(USEDIRECTIO=yes);
  ... more SAS statements ...
run;
data test.file2;
  ... more SAS statements ...
run;
```

次の例では、DATA ステートメントに列挙したファイルすべてが、128k ブロックのデータをまとめて読み込んでいます。これは、すべてのファイルが、ENABLEDIRECTIO、USEDIRECTIO=、および TRANSFERSIZE オプションにより影響を受けるためです。

```
LIBNAME test'. 'ENABLEDIRECTIO USEDIRECTIO=yes TRANSFERSIZE=128k;
data test.file1;
  ... more SAS statements ...
run;
data test.file2;
  ... more SAS statements ...
run;
data test.file3;
  ... more SAS statements ...
run;
```

**USEDIRECTIO= YES | NO**

ENABLEDIRECTIO ステートメントオプションと併用することにより、LIBNAME ステートメントにリストされているライブラリ参照名に関連付けられたすべてのファイルに対してダイレクトファイル I/O を有効化または無効化にします。

**制限事項** USEDIRECTIO を使用する場合は、Work ライブラリではなく、永久ライブラリを使用する必要があります。USEDIRECTIO を Work ディレクトリで使用することはできません。



要件 USEDIRECTIO=を ENABLEDIRECTIO ステートメントオプションと併用して、ダイレクトファイル I/O を有効化します。

## 詳細

### エンジンの種類

エンジンは、主に 2 つの種類があります。

#### ビューエンジン

SAS/ACCESS ソフトウェア、SQL プロシジャおよび DATA ステップビューによって記述される SAS ビューを、SAS で読み込めるようにします。SAS ビューの名前はデータセットのディスクリプタ部分の一部として保存されるため、SAS ビューエンジンの使用は自動です。

#### ライブラリエンジン

SAS ライブラリレベルでアクセスを制御します。すべての SAS ライブラリはライブラリエンジンが関連付けられており、そのライブラリ内のファイルは、そのエンジンを介してのみアクセスできます。ライブラリエンジンには 2 つの種類があります。

#### ネイティブエンジン

SAS によって作成され保持される SAS ファイルにアクセスします。これらのエンジンに関する説明は、次の表を参照してください。

#### インターフェイスエンジン

他のベンダのファイルを SAS ファイルと同様に処理します。

表 2.7 エンジン名と説明

エンジンの種類	名前(エイリアス)	説明	SAS ライブラリ
デフォルト	V9 (BASE) V8	SAS データファイルを新規に作成し、Version 8 または SAS 9 で作成した、既存の SAS データファイルにアクセスできるようにします。V8 と V9 のエンジンは同一です。このエンジンは、SAS の以前のリリースで作成されたデータファイルへの読み込みアクセスを可能にします。ただしこのエンジンは、SAS 9 のカタログをサポートしている唯一のエンジンです。このエンジンは、データセットのインデックス作成と圧縮を可能にします。	ライブラリが含まれているディレクトリのパス名です。
互換性	V6	6.09 - 6.12 のリリースで作成されたデータファイルにアクセスします。このエンジンは読み取り専用です。	ライブラリが含まれているディレクトリのパス名です。

エンジンの種類	名前(エイリアス)	説明	SAS ライブラリ
transport	XPORT	移送データセットにアクセスします。このエンジンは、リリース 6.06 以降の SAS を実行しているすべてのホストのもとで使用できる、コンピュータに依存しない SAS 移送ファイルを作成します。に載っています。	順次デバイスまたはディスクファイルのパス名です。
XML	XML	アプリケーションおよびコンピュータに依存しないファイルである XML ドキュメントを、生成(書き込み)したり処理(読み込み)したりします。	XML ドキュメントのパス名です。
インターフェイス	BMDP	BMDP ファイルへの読み込み専用アクセスを提供します。このエンジンは、AIX、HP-UX および Solaris でのみ使用できます。	データファイルのパス名です。
	OSIRIS	OSIRIS ファイルへの読み込み専用アクセスを提供します。	データファイルのパス名です。
	SPSS	SPSS ファイルへの読み込み専用アクセスを提供します。	データファイルのパス名です。

### LIBNAME ステートメントからのエンジン名の省略

SAS にエンジンの判断を委ねるよりも、明示的にエンジン名を指定する方が効果的です。ただし、ユーザーが LIBNAME ステートメントでエンジン名を省略する場合、または環境変数をライブラリ参照名として定義する場合には、SAS が正しいエンジンを判断します。

ENGINE=システムオプションを指定すると、SAS はこのエンジン名を使うようになります。

注: ENGINE=システムオプションは、ディスク上にあるライブラリのデフォルトエンジンのみを指定します。

ENGINE=システムオプションを指定しなかった場合、SAS は所定のディレクトリにあるファイルの拡張子を判断し、次のルールを用いてエンジンを決定します。

- ライブラリ内のすべての SAS データセットが同一のエンジンによって作成された場合、ライブラリ参照名はそのエンジンを使用して割り当てられます。

注: データセットの作成に使用されたエンジンがデフォルトのエンジンとは異なる場合、ビューまたはストアプログラムを作成することはできません。

- 所定のディレクトリに SAS データセットが存在しない場合、ライブラリ参照名はデフォルトのエンジンを使用して割り当てられます。
- 複数のエンジンの SAS データセットが存在する場合は、混合した種類のエンジンが検出されたことを通知するメッセージが発行され、ライブラリ参照はデフォルトのエンジンで割り当てられます。

**SAS ライブラリにライブラリ参照名を関連付ける(形式1)**

ライブラリ参照名と SAS ライブラリの関連付けは、SAS セッション終了まで維持されるか、または他の LIBNAME ステートメントで関連付けの変更や関連付けの取り消しを実行するまで維持されます。もっとも簡単な形式の LIBNAME ステートメントでは、1 つのライブラリ参照名と SAS ライブラリの物理名のみを指定します。

```
LIBNAME libref 'SAS-library';
```

“例 1: ライブラリ参照名の割り当てと使用” (241 ページ)を参照してください。

通常はエンジンを指定する必要はありません。状況が不明な場合、ENGINE=システムオプションの設定を使用してデフォルトエンジンを指定します。ライブラリ内のすべてのデータセットが 1 つのエンジンに関連付けられている場合、そのエンジンがデフォルト値として使用されます。どちらの状況でも、ENGINE=システムオプションに他のエンジンを指定すると、デフォルト設定より優先されます。

```
LIBNAME libref engine 'SAS-library'
<options> <engine/host-options>;
```

**動作環境の情報**

LIBNAME ステートメントを使用する場合は、ホスト固有の情報が必要です。このステートメントを使用する前に、各動作環境向けの SAS ドキュメントを参照してください。

**SAS ライブラリからライブラリ参照名の関連付けを取り消す(形式2)**

SAS ライブラリからライブラリ参照名の関連付けを取り消すには、ライブラリ参照名と CLEAR オプションを指定して LIBNAME ステートメントを使用します。指定した 1 つのライブラリ参照名のみ、または現在のすべてのライブラリ参照名の関連付けを取り消すことができます。

```
LIBNAME libref CLEAR | _ALL_ CLEAR;
```

**SAS ライブラリの属性を SAS ログに書き込む(形式3)**

1 つまたは複数の SAS ライブラリの属性を SAS ログに書き込むには、LIBNAME ステートメントを使用します。libref を指定すると、1 つの SAS ライブラリの属性を書き込みます。\_ALL\_ を使用すると、現在の SAS セッションにある割り当て済みのライブラリ参照名を含むすべての SAS ライブラリの属性を書き込みます。

```
LIBNAME libref LIST | _ALL_ LIST;
```

**SAS ライブラリの連結(形式4)**

2 つ以上の SAS ライブラリを論理的に連結すると、1 つのライブラリ参照名を使用してすべてのライブラリを参照することができます。ライブラリは、物理ファイル名またはすでに割り当て済みのライブラリ参照名を使用して指定できます。

```
LIBNAME libref <engine> (library-specification-1 <...library-specification-
n> )
< options>;
```

同一の LIBNAME ステートメントでは、ライブラリ参照名、物理ファイル名、またはライブラリ参照名と物理ファイル名の組み合わせなど、指定を組み合わせで使用できます。“例 2: SAS ライブラリの論理的連結” (241 ページ)を参照してください。

**SAS カタログの連結(形式4)**

2つ以上の SAS ライブラリを連結するときに、同じ名前の SAS カタログを連結することもできます。たとえば、3つの SAS ライブラリに CATALOG1 というカタログがそれぞれ含まれている場合、同じ名前のカタログに対してカタログ連結を作成できます。“例 3: SAS カタログの連結” (242 ページ)を参照してください。

```
LIBNAME libref <engine> (library-specification-1 <...library-specification-
n> )
< options >;
```

**ライブラリ連結のルール**

ライブラリ連結を作成すると、単純な(連結されていない)ライブラリ参照名を使用できるコンテキストであればライブラリ参照名を指定できます。連結ライブラリ中の SAS ファイル(SAS ライブラリのメンバ)の処理は、次のルールに従いません。

- SAS ファイルが入力または更新のために開かれた場合は、連結ライブラリ内  
が検索され、指定されたファイル名で最初に検出されたファイルが使用され  
ます。
- SAS ファイルが出力のために開かれた場合は、連結ライブラリの中で 1 番目  
のライブラリにファイルが作成されます。  
注: 連結の他の部分に同じ名前のファイルが存在する場合でも、新しい SAS  
ファイルは最初のカタログの中に作成されます。
- SAS ファイルの削除または名前の変更をする場合は、最初に検出されたファ  
イルが対象になります。
- SAS ファイルのリストを表示する場合は、常に 1 つのファイル名のみが表示  
されます。  
注: 連結内に同じ名前が複数存在する場合でも、最初に検索されたエントリ  
のみが表示されます。
- 他のファイルに論理的に連結された SAS ファイル(データセットのインデッ  
クスなど)は、同じライブラリ内に親ファイルが存在する場合にのみリストさ  
れます。たとえば、ライブラリ ONE に A.DATA があり、ライブラリ TWO に  
A.DATA と A.INDEX がある場合、ライブラリ ONE の A.DATA のみがリストさ  
れます。(前述の規則を参照してください)。
- 連結内に順次ライブラリがある場合、すべてのライブラリが順次ライブラリ  
として扱われます。
- 連結ライブラリに指定した最初のライブラリの属性によって、連結ライブラ  
リの属性が確定します。たとえば、指定した最初の SAS ライブラリが読み取  
り専用の場合は、連結ライブラリ全体が読み取り専用になります。
- オプションまたはエンジンを指定する場合、完全物理名で指定したライブラ  
リにのみ指定内容が適用されます。ライブラリ参照名で指定したライブラリ  
には指定内容は適用されません。
- 連結に割り当てた後にライブラリ参照名を変更しても、この変更は連結に影響  
しません。

**ライブラリディレクトリを自動的に作成する**

DLCREATEDIR システムオプションを設定すると、ディレクトリが存在しない場合、LIBNAME ステートメントに指定した SAS ライブラリのディレクトリを作成できます。詳細については、“[DLCREATEDIR System Option](#)” (*SAS Viya System Options: Reference*)を参照してください。

## 比較

- SAS ライブラリを参照するには、LIBNAME ステートメントを使用します。外部ファイルを参照するには、FILENAME ステートメントを使用します。DBMS テーブルにアクセスするには、LIBNAME、SAS/ACCESS ステートメントを使用します。
- SAS カタログを連結するには、CATNAME ステートメントを使用します。LIBNAME ステートメントを使用しても、SAS カタログを連結します。CATNAME ステートメントでは、連結するカタログの名前を指定することができます。ただし、LIBNAME ステートメントでは、指定した SAS ライブラリ内にある同じ名前のカタログをすべて連結します。

## 例

### 例 1: ライブラリ参照名の割り当てと使用

次の例では、ライブラリ参照名 SALES を集約記憶域に割り当てます。集約記憶域は、物理ファイル名を一重引用符で囲んで指定します。DATA ステップで SALES.QUARTER1 が作成され、指定した場所に格納されます。PROC PRINT ステップでは、2 レベル名 SALES.QUARTER1 を使用してライブラリ参照名を参照します。

```
libname sales 'SAS-library';
data sales.quarter1;
infile 'your-input-file';
input salesrep $20. +6 jansales febsales
      marsales;
run;
proc print data=sales.quarter1;
run;
```

### 例 2: SAS ライブラリの論理的連結

- 次の例では、各ライブラリの物理ファイル名を指定して、3 つの SAS ライブラリを連結します。

```
libname allmine ('file-1' 'file-2' 'file-3');
```

- この例では、2 つの SAS ライブラリにライブラリ参照名を割り当てます。1 つには SAS6 のファイルが含まれています。もう 1 つには SAS9 のファイルが含まれています。この処理は、両方のファイルセットにアクセスできるようになるため、ファイルとアプリケーションを SAS6 から SAS9 に更新する場合に便利です。

```
libname v6 'v6-SAS-library';
libname v9 'v9-SAS-library';
libname allmine (v9 v6);
```

- この例は、同一の連結にライブラリ参照名と物理ファイル名の両方を指定できることを示しています。

```
libname allmine (v9 v6 'some-filename');
```

**例 3: SAS カタログの連結**

この例では、各ライブラリの物理ファイル名を指定して、3つの SAS ライブラリを連結します。次に、ライブラリ参照名 ALLMINE を連結したライブラリに割り当てます。

```
libname allmine ('file-1' 'file-2' 'file-3');
```

各ライブラリに MYCAT という SAS カタログが含まれている場合、libref.catref(ライブラリ参照名.カタログ参照名)の形式で ALLMINE.MYCAT と指定すると、MYCAT という名前の3つのカタログすべてに格納されているカタログエントリにアクセスできます。異なる名前を使用して SAS カタログを論理的に連結するには、“[CATNAME ステートメント](#)” (40 ページ)を参照してください。

**例 4: 1 レベル名を使用したデータセットの永久保存**

一時ファイルではなく、永久 SAS ファイルに 1 レベル名のみを指定する場合、USER=システムオプションを使用すると便利です。この例では、格納場所にライブラリ参照名を割り当てるために、最初に LIBNAME ステートメントを使用せずに、データセット QUARTER1 を永久保存します。

```
options user='SAS-library';
data quarter1;
infile 'your-input-file';
input salesrep $20. +6 jansales febsales
      marsales;
run;
proc print data=quarter1;
run;
```

**関連項目:****データセットオプション:**

- [“ENCODING= Data Set Option” \(SAS Viya National Language Support: Reference Guide\)](#)

**ステートメント:**

- [“CATNAME ステートメント” \(40 ページ\)](#)(SAS カタログの連結の説明)
- [“FILENAME ステートメント” \(96 ページ\)](#)
- [“LIBNAME ステートメント, CVP Engine” \(243 ページ\)](#)
- [SAS ファイルのトランスコードで使用される LIBNAME オプションの文字変数属性](#)
- [“LIBNAME Statement” \(SAS/CONNECT for SAS Viya: User’s Guide\)](#)
- [“LIBNAME Statement, SASOOCK Engine” \(SAS/CONNECT for SAS Viya: User’s Guide\)](#)
- [“LOCKDOWN Statement” in SAS Intelligence Platform: Application Server Administration Guide](#)

**システムオプション:**

- [“DLCREATEDIR System Option” \(SAS Viya System Options: Reference\)](#)

- “LOCKDOWN System Option in SAS Intelligence Platform: Application Server Administration Guide
- “FILELOCKS System Option” ([SAS Viya System Options: Reference](#))
- “USER= System Option” ([SAS Viya System Options: Reference](#))

---

## LIBNAME ステートメント, CVP Engine

ファイルのトランスコードが必要なときに文字データの切り捨てが発生しなくなるように、文字変数の長さを拡張する文字変数埋め込み(CVP)エンジンのライブラリ参照名を関連付けます。

**該当要素:** 任意の場所

**カテゴリ:** データアクセス

**制限事項:** このステートメントは CAS Server ではサポートされていません。

CVP エンジンは入力(読み取り)処理でのみ使用できます。

CVP エンジンは SAS データファイルをサポートしていますが、SAS のビュー、カタログ、アイテムストアはサポートしていません。

指定するバイト数は、どのような拡張にも対応できる大きさにする必要があります。そうではない場合には切り捨てが発生して SAS ログにエラーメッセージが書き込まれます。

CVP エンジンを含むエンジンの混在したライブラリ連結では、SAS データファイルのみが処理されます。たとえば、COPY プロシジャを実行すると SAS データファイルのみがコピーされます。

**ヒント:** CVP エンジンは、文字変数の長さは拡張しますが文字出力形式の幅は拡張しません。幅が十分でない文字出力形式を使うと、オブザベーションを正しく印字できない場合があります。

---

### 構文

```
LIBNAME libref CVP 'SAS-library' <options>;
```

### 引数

#### *libref*

SAS ライブラリに割り当てるライブラリ参照名を示す文字定数、変数、式を指定します。

範囲 1~8 バイト

#### *SAS-library*

SAS ライブラリの物理名を指定します。物理名が動作環境で認識されることが必要です。物理名は一重引用符または二重引用符で囲みます。

### LIBNAME オプション

#### CVPBYTES=*bytes*

トランスコーディングを必要とする SAS データファイルを処理するときに、拡張する文字変数の長さをバイト数で指定します。CVP エンジンは、長さを拡張することにより、文字データの切り捨てが発生しないようにします。文字変数の長さは、指定した値を現在の長さに追加して増やします。0 から 32766 までの値を指定できます。



たとえば、次の LIBNAME ステートメントでは、CVPBYTES=オプションを指定して CVP エンジンを暗黙的に割り当てています。

```
libname expand 'SAS data-library' cvpbytes=5;
```

文字変数の長さは、5 バイト追加して増やします。長さが 10 の文字変数は長さ 15 に、長さが 100 の文字変数は長さ 105 に増加します。

**デフォルト** CVPBYTES=を指定すると、指定値に従って文字変数の長さを拡張するために CVP エンジンを自動的に使用します。CVP エンジンを明示的に割り当てても、CVPBYTES=または CVPMULTIPLIER=を指定しない場合は、CVPMULTIPLIER=1.5 を使用して文字変数を長くします。

**制限事項** CVP エンジンは、SAS データファイルをサポートしていますが、SAS のビュー、カタログおよびアイテムストアなどはサポートしていません。

CVP エンジンは、入力(読み込み)処理でのみ使用できます。

CVP エンジンなどのエンジンが混在するライブラリ連結では、SAS データファイルのみが処理されます。たとえば、COPY プロシジャを実行する場合は、SAS データファイルのみコピーされます。

**要件** 指定するバイト数は、どのような拡張にも対応できる大きさにする必要があります。この大きさが不十分だと、データが切り捨てられ、その結果 SAS ログにエラーメッセージが書き込まれます。

**操作** CVPBYTES=と CVPMULTIPLIER=の両方を指定できません。いずれか 1 つのオプションを指定してください。

**参照項目:** [“Avoiding Character Data Truncation By Using the CVP Engine” \(SAS Viya National Language Support: Reference Guide\)](#)

### **CVPENGINE=engine**

トランスコーディングを必要とする SAS データファイルを処理するために使用するエンジンを指定します。トランスコードする文字変数の長さが CVP エンジンによって拡張されるため、文字データの切り捨ては発生しません。次に、指定されたエンジンで実際のファイル処理を実行します。

**別名** CVPENG

**デフォルト** SAS は、デフォルトの SAS エンジンを使用します。

**参照項目:** [“Avoiding Character Data Truncation By Using the CVP Engine” \(SAS Viya National Language Support: Reference Guide\)](#)

### **CVPMULTIPLIER=multiplier**

トランスコーディングを必要とする SAS データファイルを処理するときに、文字変数の長さを拡張するための乗数値を指定します。CVP エンジンは、長さを拡張することにより、文字データの切り捨てが発生しないようにします。文字変数の長さは、指定した値を現在の長さに乗算して増やします。1 から 5 までの乗数値を指定できます。

たとえば、次の LIBNAME ステートメントでは、CVPMULTIPLIER=オプションを指定して CVP エンジンを暗黙的に割り当てています。

```
libname expand 'SAS data-library' cvpmultiplier=2.5;
```

文字変数の長さは 2.5 倍になります。長さが 10 の文字変数は長さ 25 に、長さが 100 の文字変数は長さ 250 に増加します。



別名	CVPMULT
デフォルト	CVPMULTIPLIER=を指定すると、指定値に従って文字変数の長さを拡張するために CVP エンジンを使用します。CVP エンジンを実示的に割り当てても、CVPMULTIPLIER=またはCVPBYTES=を指定しない場合は、CVPMULTIPLIER=1.5 を使用して文字変数を長くします。
制限事項	CVP エンジンは、SAS データファイルをサポートしていますが、SAS のビュー、カタログおよびアイテムストアなどはサポートしていません。  CVP エンジンは、入力(読み込み)処理でのみ使用できます。  CVP エンジンなどのエンジンが混在するライブラリ連結では、SAS データファイルのみが処理されます。たとえば、COPY プロシジャを実行する場合は、SAS データファイルのみコピーされます。
要件	指定するバイト数は、どのような拡張にも対応できる大きさにする必要があります。この大きさが不十分だと、データが切り捨てられ、その結果 SAS ログにエラーが書き込まれます。
操作	CVPMULTIPLIER=と CVPBYTES=の両方を指定できません。いずれか1つのオプションを指定してください。
参照項目:	<a href="#">“Avoiding Character Data Truncation By Using the CVP Engine” (SAS Viya National Language Support: Reference Guide)</a>

**CVPVARCHAR=YES | NO**

入力ファイル処理の際に、固定幅文字変数を可変幅文字に変換するか否かを指定します。固定幅文字変数のバイト長は、SAS セッションエンコーディングにおける 1 文字当たり最大バイト数に、指定された固定幅文字長を掛けた値です。

デフォルト	No
操作	CVPVARCHAR=YES を指定した場合、CVPMULTIPLIER=オプションや CVPBYTES=オプションを指定しても無視されます。
注	CHAR カラム以下にある文字列データの末尾に空白があれば削除します。  TRANSCODE=NO という形式の固定幅文字変数は、変換の対象外です。

**詳細**

ファイルのトランスコードが必要な場合に文字データの切り捨てが発生しなくなるように、文字変数埋め込み(CVP)エンジンにより文字変数の長さが拡張されます。文字データの切り捨ては、あるエンコーディングにおける文字のバイト数が、別のエンコーディングにおける同じ文字のバイト数と異なる場合に発生します。例としては、単一バイト文字セット(SBCS)を 2 バイト文字セット(DBCS)またはマルチバイト文字セット(MBCS)にトランスコードした場合が挙げられます。

LIBNAME=ステートメントによって CVP エンジン<sup>1</sup>を明示的に指定した場合、デフォルトの文字拡張では文字変数の長さが 1.5 倍になります。別の拡張量を指定するには、CVPBYTES=または CVPMULTIPLIER=オプションを使用します。

注 拡張量は、どのような拡張にも適応できる大きさにする必要があります。そうしない場合は、切り捨てが発生します。

CVPVARCHAR=YES を指定した場合、CVP エンジンは、CHAR データタイプで定義された変数を VARCHAR データタイプへと変換します。CVP は入力処理でのみ利用できるため、VARCHAR データタイプは自動的に保存されません。データを保存しない場合、VARCHAR への変換は、SAS セッションが終了した時点で失われます。変更を保存するには、SET ステートメントまたは PROC COPY と、VARCHAR をサポートするエンジンを使用します。VARCHAR をサポートしていないエンジンを使用してデータを保存した場合、新しいデータセット内の文字カラムは CHAR データタイプへと戻ります。新しいデータセット内の CHAR 変数の長さは、VARCHAR 変数を格納するために必要となるバイト数へと設定されません。

VARCHAR をサポートしていないプロシジャによりデータが読み取られる場合、CVPVARCHAR=YES は無視されます。CVP エンジンは、CVPBYTES、CVPMULTIPLIER、またはデフォルトの乗数を使用して、データ内の文字カラムの長さを拡張します。

**ヒント** CVP エンジンは、WHERE 式を使用してオブザベーションのサブセットを条件的に選択する処理のパフォーマンスに影響を与えます。CVP エンジンを使用しないファイルの処理は、CVP エンジンを使用したファイルの処理より高速になる場合があります。たとえば、インデックスを含むデータセットで CVP を使用する場合、WHERE 式を最適化するために、それらのインデックスは使用されません。

## 例

### 例 1

次の LIBNAME ステートメントでは明示的に CVP エンジンを割り当てています。文字変数の長さは、長さに 1.5 を乗算するデフォルトの拡張を使用して増やされています。たとえば、長さが 10 の文字変数の新しい長さは 15 になります。長さが 100 の文字変数の新しい長さは 150 になります：

```
libname expand cvp 'SAS-library';
```

### 例 2

この例では、それぞれの長さに 2 バイトを追加して文字変数の長さを拡張するために CVP エンジンが使用されます：

```
libname expand cvp 'SAS-library' cvpbytes=2;
```

### 例 3

この例では、データセットファイルにアクセスするために SPD エンジンが使用されます。データセットを読み取る時、たとえば PROC CONTENTS または PROC PRINT ステップの実行中に、CHAR 変数を VARCHAR 変数に変換するために CVP エンジンが使用されます。

```
libname cvplib cvp 'SAS-library' cvpvvarchar=yes cvpengine=spde;
```

**例 4**

次の例では、CVP エンジンを使用して CHAR 変数を VARCHAR に変換する方法を示します。TRANSCODE=NO 属性を使用して、変数 ch2 はこの変換から除外されます。

```
libname baselib 'c:\temp\testdata';
libname cvplib cvp 'c:\temp\testdata' cvpvvarchar=yes;

data baselib.latin1;
    length ch1 $ 10 ch2 $ 20;
    attrib ch1 transcode=yes;
    attrib ch2 transcode=no;
    ch1='hello';
    ch2='hello world';

run;

proc contents data=cvplib.latin1;
run;
```

**関連項目:**

- [“Avoiding Character Data Truncation By Using the CVP Engine” \(SAS Viya National Language Support: Reference Guide\)](#)

**ステートメント:**

- [“LIBNAME ステートメント” \(225 ページ\)](#)

---

**LIBNAME ステートメント, JMP Engine**

ライブラリ参照名を JPM データテーブルに関連付け、JMP データテーブルの読み込みや書き込みを実行できるようにします。

- 該当要素:** 任意の場所
- カテゴリ:** データアクセス
- 制限事項:** このステートメントは CAS Server ではサポートされていません。
- 参照項目:** [“LIBNAME ステートメント” \(225 ページ\)](#)
- 

**構文**

```
LIBNAME libref JMP 'path' <FMTLIB=libref.format-catalog>;
```

**引数*****libref***

SAS ライブラリに割り当てるライブラリ参照名を示す文字定数、変数、式を指定します。

範囲 1~8 バイト

---

***path***

SAS ライブラリの物理名を指定します。物理名には動作環境で判別できる名前を指定します。物理名は一重引用符または二重引用符で囲みます。

**FMTLIB=libref.format-catalog**

JMP データテーブル読み込み時にフォーマットが格納される場所、および JMP データテーブル作成時にフォーマットを取得する場所を指定します。

**要件** この FMTLIB 引数に指定するライブラリは、SAS データセットのライブラリ参照名の必要があります。

**例**

```
libname inv jmp "." fmtlib=seform.formats;
libname seform '!';
data work.mine;
set inv.suri2011;
run;
```

**詳細**

JMP ファイルとは、JMP ソフトウェアが作成するファイル形式です。JMP とは、Microsoft Windows および Macintosh 上で使用できる対話型の統計パッケージソフトウェアです。詳細については、システムに同梱されている JMP のドキュメントを参照してください。

JMP ファイルには、フィールドとレコードが表形式でまとめられているデータが含まれます。各フィールドには 1 種類のデータを含めることができます。各レコードには各フィールドのデータ値を 1 つ含めることができます。

SAS では、JMP ファイルへのアクセスをサポートしています。次の 2 つの方法のいずれかを使用して、JMP ファイルにアクセスできます。

- IMPORT/EXPORT プロシジャ、インポート/エクスポートウィザード (SAS/ACCESS Interface to PC Files のライセンスを使用しない場合)。SAS はバージョン 7 以降の形式で保存された JMP ファイルをインポートして、バージョン 7 以降の形式の JMP ファイルにデータをエクスポートします。バージョン 3 から 6 形式の JMP ファイルは、現在サポートされていません。

詳細については、[SAS/ACCESS Interface to PC Files for SAS Viya: Reference](#) を参照してください。

- JMP Engine の LIBNAME ステートメント

**注** JMP LIBNAME Engine は拡張属性をサポートしません。拡張属性が必要な場合、**dbms=jmp** オプションを指定した IMPORT プロシジャまたは EXPORT プロシジャを使用します。

**例****例 1: LIBNAME ステートメントを使用して JMP データテーブルを読み込む**

この例では、JMP データテーブル **bank** から 5 つのオブザベーションの読み込みと出力を行います。

```
libname b jmp 'c:/temp/national';
proc contents data=b.bank(drop=edlevel id age);
run;
proc print data=b.bank(obs=5 drop=edlevel id age);
run;
```

**例 2: JMP データテーブルの読み込みと並べ替え**

この例では、JMP データテーブルを読み込んでから並べ替えを行い、SAS データセットに格納します。データセットに格納されているフォーマットは、**a.formats** にあります。

```
libname a 'c:/temp/field';
libname b jmp '!' fmlib=a.formats;

proc sort data=b.cars out=a.sorted;
  by category_ic;
run;
```

---

**LIBNAME ステートメント, JSON Engine**

SAS ライブラリ参照名を JSON ドキュメントと関連付けて、JSON データと、指定された任意の MAP が有効になることを保証します。

**該当要素:** 任意の場所

**カテゴリ:** データアクセス

**制限事項:** このステートメントは CAS Server ではサポートされていません。

**注:** JSON エンジン LIBNAME が割り当てられている場合、JSON ファイルは 1 回のみ読み取られます。JSON ファイルを再度読み取るには、JSON LIBNAME ライブラリ参照名を再度割り当てる必要があります。“[例 2: URL アクセス方式ファイル参照名からの JSON の使用法](#)” (262 ページ) を参照してください。

---

**構文**

```
LIBNAME libref JSON < 'JSON-document-path' > <options>;
```

**引数*****libref***

SAS ライブラリに割り当てる論理名またはライブラリ参照名を示す文字定数、変数、式を指定します。

範囲 1~8 バイト

***JSON-document-path***

JSON ドキュメントの物理的な場所を指定します。物理的な場所は、一重または二重引用符で囲みます。

注: 物理的な場所を指定しない場合、JSON エンジンは、指定されたライブラリ参照名と同じ名前を持つファイル参照名にアクセスしようとします。ファイル参照名としてディスクファイル、URL アクセス方式ファイル参照名、FTP アクセス方式ファイル参照名またはその他の有効なファイル参照名を指定できます。

**LIBNAME オプション****AUTOMAP= REUSE | CREATE**

実行するアクションを指定します。AUTOMAP は次のいずれかの値を持ちます。

**REUSE**

ファイルが存在する場合は、MAP ファイル参照名が使用されます。存在しない場合は、JSON マップが生成され、MAP ファイル参照名に書き込まれます。

**CREATE**

JSON マップが生成され、MAP ファイル参照名に書き込まれます。

注: REPLACE または CREATE を使用すると、既存のファイルが上書きされます。

別名 REPLACE

要件 JSON マップファイルを指定したときに、それが存在しない場合には、AUTOMAP オプションが必要です。

操作 JSON マップファイルを指定しない場合、JSON マップファイルが自動的にメモリ内に作成されるため、AUTOMAP オプションは必要ありません。

**FILEREF=*fileref***

JSON 入力の場所を指すファイル参照名を指定します。

**JSONCOMPRESS | JSONNOCOMPRESS**

内部 JSON 情報を圧縮するかどうかを指定します。

**ヒント** 情報をキャッシュするときに、圧縮すると CPU 時間は消費されますが、メモリとディスクスペースは節約されます。

デフォルト JSONNOCOMPRESS

**MAP=*fileref* | '*map physical name*'**

specifies a fileref that points to a physical location for the JSON map, or the quoted physical name of the JSON map file.

**MEMLEAVE= *n* | *nK* | *nM* | *nG* | ALL**

JSON の処理時に利用可能な状態にするメモリ容量を指定します。メモリの制限に到達すると、内部的に生成される TEMP ファイル参照名を使用して、JSON 情報はディスクにキャッシュされます。

*n*

メモリ容量

*nK*

メモリ容量をキロバイト単位で指定します。

*nM*

メモリ容量をメガバイト単位で指定します。

*nG*

メモリ容量をギガバイト単位で指定します。

ALL

JSON 情報を保存するためのメモリを使用しないことを指定します。すべての JSON 情報はディスクにキャッシュされます。

デフォルト 0.5G

**NO\_REOPEN\_MSG | NRM**

エラーメッセージは非表示にされ、JSON データは 1 回のみ読み取られます。JSON を再度読み取るには、JSON LIBNAME を再度割り当てます。

注 JSON エンジン LIBNAME が割り当てられている場合、JSON ファイルは 1 回読み取られます。JSON ファイルを再度読み取るには、LIBNAME ライブラリ参照名の割り当てを解除し、再度割り当てる必要があります。  
[“例 2: URL アクセス方式ファイル参照名からの JSON の使用法” \(262 ページ\)](#)を参照してください。

#### **ORDINALCOUNT= ALL | NONE | *n***

各データセット用に生成する順序変数の最大数を指定します。1 つのデータセットについて可能な順序変数の最大数は、JSON 内のデータの位置によって異なり、ユーザーが指定する数より小さくなる場合があります。その場合には、可能な順序変数の最大数のみが生成されます。ORDINALCOUNT は次のいずれかの値を持ちます。

##### **ALL**

データセットについて可能な順序変数がすべて作成されます。

##### **NONE**

データセットの順序変数の作成が抑制されます。

##### ***n***

0(ゼロ)以上の整数を指定します。

デフォルト 2

#### **RETAIN**

オブザベーションの間にオブザベーションバッファ内に値が保持されます。

## 詳細

### 基本

JavaScript Object Notation (JSON)ファイルは、人間が読めるデータのコレクションを含むファイルです。

JSON LIBNAME ステートメントを使用すると、JSON エンジンを使用して JSON ファイルにアクセスできます。JSON マップは、JSON エンジンライブラリ内のデータセットを記述するファイルです。

### JSON MAP の詳細

JSON マップは、JSON エンジンライブラリ内のデータセットを記述するファイルです。指定した場合、JSON オートマップは自動的に JSON データのマップを生成できます。

JSON マップは DATASETS 配列から構成されます。

DATASETS 配列内の各オブジェクトによりライブラリ内のデータセットが記述されます。

各データオブジェクトには DSNAME 文字列と TABLEPATH 文字列が必要で、通常は任意の VARIABLES 配列を含んでいます。

VARIABLES 配列内の各オブジェクトにより、そのデータセット内の変数が記述されます。

各 VARIABLES オブジェクトに必要な項目は、NAME 文字列、TYPE 文字列および PATH 文字列です。

文字 TYPE 変数の場合、LENGTH 項目は任意です。

**DSNAME**

DSNAME によりデータセットの名前を指定します。オートマップによりデフォルトの名前が生成されます。ユーザーのマップファイルを変更する場合は、SAS データセットの命名基準に従う任意の名前を指定できます。

**TABLEPATH**

TABLEPATH パスにより、データセットオブザベーションを区切る方法がエンジンに伝えられます。

```
{
  "data": [
    { "a": 1, "b": 2, "c": "taxes" },
    { "a": 2, "b": 4, "c": "sport" },
    { "a": 3, "b": 6, "c": "vacation" }
  ]
}
```

JSON では、データ配列内でオブジェクトに遭遇するたびにオブザベーションが必要な場合には、TABLEPATH を "/root/data" と設定します。JSON マップにより自動的に次の JSON マップが生成されます。

```
{
  "DATASETS": [
    {
      "DSNAME": "data",
      "TABLEPATH": "/root/data",
      "VARIABLES": [
        {
          "NAME": "ordinal_root",
          "TYPE": "ORDINAL",
          "PATH": "/root"
        },
        {
          "NAME": "ordinal_data",
          "TYPE": "ORDINAL",
          "PATH": "/root/data"
        },
        {
          "NAME": "a",
          "TYPE": "NUMERIC",
          "PATH": "/root/data/a"
        },
        {
          "NAME": "b",
          "TYPE": "NUMERIC",
          "PATH": "/root/data/b"
        },
        {
          "NAME": "c",
          "TYPE": "CHARACTER",
          "PATH": "/root/data/c",
          "CURRENT_LENGTH": 8
        }
      ]
    }
  ]
}
```



**NAME**

NAME により、SAS 命名基準に従う変数の名前を指定します。

**TYPE**

TYPE は ORDINAL、NUMERIC または CHARACTER です。

**PATH**

PATH はその変数の JSON マップファイル内のパスです。

**LENGTH**

LENGTH 設定は、その文字変数の任意の長さです。長さを指定しない場合、JSON LIBNAME エンジン は、その変数について参照したすべての値の最大長をその長さとして設定します。

注 別の JSON 入力、特により長い文字列を含む JSON 入力を使用してマップを再利用する場合、マップ LENGTH は効果を保ち続けるため、切り捨てが発生する場合があります。

ヒント JSON LIBNAME エンジンによりオートマップが生成される場合には、CURRENT\_LENGTH: n の値が生成され、現在の JSON データ内の変数の最大の長さが設定されます。これはドキュメンテーションのためのみ行われます。マップを読み取る場合、JSON LIBNAME エンジンにより CURRENT\_LENGTH 設定は無視されます。

**FORMAT**

マップ内の変数に FORMAT を指定できます。FORMAT は配列で、配列の最初の要素は形式の名前、2 番目の要素は幅、3 番目の要素は小数の指定です。FORMAT 変数は任意です。

```
"FORMAT" : [ "FormatName", "width", "decimal-specification" ]
```

注 現在、JSON LIBNAME エンジンオートマップにより FORMAT は生成されません。

例 次のマップにより小数点以下 3 桁を含む合計 12 桁の形式 BEST が指定されます。

```
{
  "NAME": "c",
  "TYPE": "NUMERIC",
  "PATH": "/root/nums/c",
  "FORMAT" : [ "BEST", 12, 3 ]
}
```

**INFORMAT**

マップ内の変数に INFORMAT を指定できます。INFORMAT は配列で、配列の最初の要素は入力形式の名前、2 番目の要素は幅、3 番目の要素は小数の指定です。INFORMAT 変数は任意です。

```
"INFORMAT" : [ "InformatName", "width", "decimal-specification" ]
```

注 現在、JSON LIBNAME エンジンオートマップにより INFORMAT は生成されません。

例 次のマップにより、小数位を含まない 19 桁を含む入力形式 IS8601DT が指定されます。

```
{
  "NAME": "d",
  "TYPE": "NUMERIC",
  "PATH": "/root/nums/d",
  "INFORMAT" : [ "IS8601DT", 19, 0 ]
}
```

```
}

```

## LABEL

マップ内の変数に LABEL を指定できます。LABEL 変数は任意です。

```
"LABEL" : [ "text" ]
```

注 現在、JSON LIBNAME エンジンオートマップにより LABEL は生成されません。

例 次のマップにより、変数 d のラベルが指定されます。

```
{
  "NAME": "d",
  "LABEL": "This is the d variable.",
  "TYPE": "NUMERIC",
  "PATH": "/root/nums/d"
}
```

## OPTIONS

個別の変数について RETAIN のオンとオフを切り替えることができます。OPTIONS 変数は任意です。詳細については、[“RETAIN” \(251 ページ\)](#)を参照してください。

```
"OPTIONS" : [ "RETAIN" | "NORETAIN" ]
```

注 ORDINAL 変数に NORETAIN を設定しても効果はありませんが、マップの検証中に NOTE が生成されます。ORDINAL 変数は常に保持されます。

ヒント マップ内の変数に RETAIN を設定すると、LIBNAME ステートメントの RETAIN オプションより優先されます。たとえば、LIBNAME ステートメント内で RETAIN を設定し、JSON 変数 Status に NORETAIN を設定すると、Status を除くすべての変数が保持されます。

例 変数 Status に対する RETAIN の有効化:

```
{
  "NAME": "Status",
  "TYPE": "CHARACTER",
  "PATH": "/root/info/Status",
  "OPTIONS" : ["RETAIN"]
}
```

## ユーザー独自の JSON MAP の作成

この例では、すべてのタイプ、名前および年齢情報を含む従業員電話番号のリストを、JSON マップを使用して取得する方法を示します。最初のステップでは、次の JSON ファイル example.json から JSON マップを作成します。

```
[
  {
    "type": "Full",
    "info": [
      { "name": "Eric", "age": 21, "phone": [
        { "type": "cell", "number": "540-555-2377" },
        { "type": "home", "number": "540-555-0120" }
      ]
    }
  ]
}
```

```

    },
    { "name": "John", "age": 22, "phone": [
      { "type": "cell", "number": "919-555-6665" },
      { "type": "home", "number": "336-555-0140" }
    ]
  }
]
},
{
  "type": "Part",
  "info": [
    { "name": "Bjorn", "age": 27, "phone": [
      { "type": "cell", "number": "720-555-8377" },
      { "type": "burner", "number": "720-555-2877" },
      { "type": "home", "number": "720-555-0194" }
    ]
  }
}
]
}
]

```

次の LIBNAME ステートメントにより JSON マップファイル user.map が生成されます。

```
libname in json 'example.json' map='user.map' automap=create;
```

JSON により、INFO、INFO\_PHONE および ROOT の 3 つのデータセットが生成されます。

3 つのデータセットの内容を出力するには、PROC PRINT を使用します。

```

6 proc print data=in.root; run; ordinal_ OBS root type 1 1 Full 2 2 Part 7 proc print data=in.info; run;
ordinal_ ordinal_ OBS root info name age 1 1 1 Eric 21 2 1 2 John 22 3 2 3 Bjorn 27 8 proc print
data=in.info_phone; run; ordinal_ ordinal_ OBS info phone type number 1 1 1 cell 540-555-2377 2 1 2
home 540-555-0120 3 2 3 cell 919-555-6665 4 2 4 home 336-555-0140 5 3 5 cell 720-555-8377 6 3 6
burner 720-555-2877 7 3 7 home 720-555-0194

```

すべての情報を含むデータセットを 1 つ作成するには、JSON オートマップにより作成される user.map ファイルを user.map.all にコピーします。次のようにして user.map.all を変更します。

- 最初のデータセットの DSNAME を ALL に変更します。
- ALL データセットから ORDINAL 変数を削除します。
- 必要な変数の説明を、他のデータセットから ALL データセットにコピーします。最後の変数オブジェクトを除く各変数オブジェクトの後ろにカンマがあることを確認します。
- タイプ変数の名前を INFO\_PHONE データセットから PHONETYPE に変更します。
- LABEL を PHONETYPE 変数に追加します。
- 電話オブジェクトに遭遇した後にオブザベーションを取得するには、TABLEPATH を /root/info/phone と設定します。
- 各変数から CURRENT\_LENGTH を削除します。

変更後のマップを次に示します。

```
{
```

```

"DATASETS": [
  {
    "DSNAME": "all",
    "TABLEPATH": "/root/info/phone",
    "VARIABLES": [
      {
        "NAME": "type",
        "TYPE": "CHARACTER",
        "PATH": "/root/type"
      },
      {
        "NAME": "name",
        "TYPE": "CHARACTER",
        "PATH": "/root/info/name"
      },
      {
        "NAME": "age",
        "TYPE": "NUMERIC",
        "PATH": "/root/info/age"
      },
      {
        "NAME": "phonetype",
        "TYPE": "CHARACTER",
        "LABEL": "This is the type of phone",
        "PATH": "/root/info/phone/type"
      },
      {
        "NAME": "number",
        "TYPE": "CHARACTER",
        "PATH": "/root/info/phone/number"
      }
    ]
  }
]
}

```

次の結果は、変更したマップと PROC PRINT を使用して表示されます。

```

10 filename allmap 'user.map.all'; 11 libname in json 'example.json' map=allmap; 12 proc
print data=in.all label; run; This is the type OBS type name age of phone number 1 Full
Eric 21 cell 540-555-2377 2 .home 540-555-0120 3 John 22 cell
919-555-6665 4 .home 336-555-0140 5 Part Bjorn 27 cell 720-555-8377
6 .burner 720-555-2877 7 .home 720-555-0194

```

Eric の携帯電話と Eric の自宅電話の間でオブザベーションバッファがクリアされていることに注意してください。オブザベーションバッファがクリアされないように維持するには、RETAIN オプションを使用します。

```

13 libname in json 'example.json' map=allmap RETAIN; 14 proc print data=in.all label; run; This is the
type OBS type name age of phone number 1 Full Eric 21 cell 540-555-2377 2 Full Eric 21 home
540-555-0120 3 Full John 22 cell 919-555-6665 4 Full John 22 home 336-555-0140 5 Part Bjorn 27 cell
720-555-8377 6 Part Bjorn 27 burner 720-555-2877 7 Part Bjorn 27 home 720-555-0194

```

### データセットのマージ

この例では、JSON を SAS データセットに読み取り、次に順序値を使用して各データセットをマージする方法を説明します。

単一のデータセットに読み取るサンプルの JSON ファイルを次に示します。2 種類のタイプ、Work と Holding があります。ステータスは、complete、in the middle または not started です。さらに、Name、Date、Age、Address、Zip の各情報があります。

```
[
  { "Type": "Work",
    "info": [{ "Status": "complete",
              "name": { "name": "Eric", "Date": "28sep15", "age": 21 },
              "add": { "Address": "55 Pelican Avenue", "zip": 44442 }
            },
            { "Status": "in the middle",
              "name": { "name": "John", "Date": "02oct15", "age": 22 },
              "add": { "Address": "268 Hydrangea", "zip": 40207 }
            }
          ]
  },
  { "Type": "Holding",
    "info": { "Status": "not started",
              "name": { "name": "Bjorn", "Date": "01dec15", "age": 27 },
              "add": { "Address": "1217 Onslow", "zip": "22801" }
            }
  }
]
```

次により、ファイル参照名 IN へと入力 JSON ドキュメントを読み取るために、JSON LIBNAME エンジンが割り当てられます。作成される 4 つのデータセットの名前とメンバタイプを表示するには、PROC DATASETS を使用します。

```
filename in 'example.json';
filename map 'example.map';
libname in json map=map automap=replace;
proc datasets library=in;
run;
quit;
```

```
1 options nocenter; 2 filename in 'example.json'; 3 filename map 'example.map'; 4 libname in json
map=map automap=replace; NOTE: JSON data is only read once.To read the JSON again, reassign the
JSON LIBNAME.NOTE: Map file /r/example.com/mydir/work/json/example.map was replaced.NOTE:
Libref IN was successfully assigned as follows: Engine: JSON Physical Name: /r/example.com/mydir/
work/json/example.json 5 proc datasets library=in; Directory Libref IN Engine JSON Access READONLY
Physical Name /r/example.com/mydir/work/json/example.json Member # Name Type 1 INFO DATA 2
INFO_ADD DATA 3 INFO_NAME DATA 4 ROOT DATA 6 run; 7 quit;
```

4 つのデータセットの内容を出力するには、PROC PRINT を使用します。

```
proc print data=in.root; run; ordinal_ Obs root Type 1 1 Work 2 2 Holding proc print data=in.info_name;
run; ordinal_ ordinal_ Obs info name name Date age 1 1 1 Eric 28sep15 21 2 2 John 02oct15 22 3 3
Bjorn 01dec15 27 proc print data=in.info; run; ordinal_ ordinal_ Obs root info Status 1 1 1 complete 2 1
2 in the middle 3 2 3 not started proc print data=in.info_add;run; ordinal_ ordinal_ Obs info add Address
zip 1 1 1 55 Pelican Avenue 44442 2 2 2 268 Hydrangea 40207 3 3 3 1217 Onslow 22801
```

4 つのデータセットには順序変数が含まれています。順序変数はキー変数で、2 つのデータセットの関係を指定します。

2 番目のデータセットの INFO\_NAME は John の名前があるオブザベーションを含み、ordinal\_info の値は 2 です。4 番目のデータセットの INFO\_ADD では、2

の ordinal\_info と関連付けられているアドレスは 268 Hydrangea です。これは John と関連付けられたアドレスです。

順序値に基づいて 4 つのデータセットをマージできます。まず、ORDINAL\_ROOT により root と info のデータセットをマージします。

```
data a; merge in.root in.info; by ORDINAL_root; run; NOTE: There were 2 observations read from the
data set IN.ROOT.NOTE: There were 3 observations read from the data set IN.INFO.NOTE: The data set
WORK.A has 3 observations and 4 variables. proc print data=a; run; ordinal_ ordinal_ Obs root Type info
Status 1 1 Work 1 complete 2 1 Work 2 in the middle 3 2 Holding 3 not started
```

次に、すべての JSON 情報を含むデータセットの作成を完了するために、name と address をマージして ORDINAL 変数をドロップします。

```
data b; merge a in.info_name in.info_add; by ORDINAL_info; drop ORDINAL_info ORDINAL_name
ORDINAL_add ORDINAL_root; run; NOTE: There were 3 observations read from the data set
WORK.A.NOTE: There were 3 observations read from the data set IN.INFO_NAME.NOTE: There were 3
observations read from the data set IN.INFO_ADD.NOTE: The data set WORK.B has 3 observations and
7 variables. proc print data=b; run; Obs Type Status name Date age Address zip 1 Work complete Eric
28sep15 21 55 Pelican Avenue 44442 2 Work in the middle John 02oct15 22 268 Hydrangea 40207 3
Holding not started Bjorn 01dec15 27 1217 Onslow 22801
```

## ALLDATA データセット

ALLDATA データセットを使用すると、1 つのデータセット内のすべての JSON データにアクセスできます。次に例を示します。

```
proc print data=in.ALLDATA(obs=24); run; OBS P P1 P2 P3 P4 V Value 1 1 stores
```

ALLDATA データセットの例には次の変数が含まれます。

- P は NUMERIC データタイプで、P1~P4 の変数のうちのいくつかがこのオブザベーションに関する情報を含んでいるかを示します。
- P1~P4 は CHARACTER データタイプです。
- V は NUMERIC データタイプで、Value が利用可能かどうかを示します。
- Value は CHARACTER データタイプであり、JSON の値です。

オブザベーション 1 内にある P=1 V=0 は、P1 に情報が含まれ、P2~P4 は空白で、Value が空白であることを示しています。オブザベーション 6 内にある P=4 は、P1~P4 の変数に情報が含まれていることを示し、V=1 が Value に値が含まれていることを示しています。

V=0 であるオブザベーションは、新しい JSON オブジェクトの始まりを示しています。たとえば、オブザベーション 1 および 18 内にある P=1 および V=1 は、新しい Stores オブジェクトが始まることを示しています。オブザベーション 5 は、新しい stores/sales/Hot\_Dogs object オブジェクトを示しています。V=0 であるオブザベーションは、JSON オブジェクトを追跡するのに役立ちます。

また、JSON LIBNAME エンジンには、ALLDATA データセットの長さを記述するマクロシンボルも作成します。

JSON サンプルのマクロ変数と値を次に示します。JSON LIBNAME は **IN** です。

```
IN_JADPNUM=4
IN_JADVLEN=20
IN_JADP1LEN=6
IN_JADP2LEN=6
IN_JADP3LEN=11
IN_JADP4LEN=5
```

**ALLDATA データセットの使用法**

各オブザベーションに次のような変数が含まれるデータセットを作成します。

- item
- 項目の価格と数
- ストア名とコード

ALLDATA データセットを参照する際、まず Store Name を参照し、次に項目名、数および価格を含むオブザベーションを参照します。ALLDATA データセットから 2 つのデータセットを作成します。一方のデータセットの名前は StoreInfo でストアの Name と Code を含みます。もう一方のデータセットの名前は ItemInfo で Item、Count および Price を含みます。Price が見つかるたびに、オブザベーションが ItemInfo データセットに書き込まれます。Code が見つかるたびに、オブザベーションが StoreInfo データセットに書き込まれます。2 つのデータセットをマージする場合、各データセットには、オブザベーションデータがどのストアに由来するかを示すキーが含まれている必要があります。サンプル用のコードを次に示します。

data

```
storeinfo (keep = Key StoreName Code)
iteminfo (keep = Key Item Price Count)
;
/*-----*/
/* bring in ALLDATA */
/*-----*/
set in.ALLDATA;

/*-----*/
/* Since StoreName and Code are found in the Value variable of ALLDATA */
/* the length of StoreName and COde should be &IN_JADVLEN Similarly, */
/* Item comes from P3, so its length is &IN_JADP3LEN. */
/*-----*/
length StoreName $ &IN_JADVLEN Code $ &IN_JADVLEN Item $ &IN_JADP3LEN;

/*-----*/
/* Both StoreName and Count are picked up in observations before the */
/* observations which tell us to write and output observation. So we */
/* need to retain StoreName and Count */
/*-----*/
retain StoreName Count;

/*-----*/
/* We want a Key which we'll increment each time we see a store object */
/* (v=0 p=1 and P1="stores" */
/*-----*/
retain Key 0;

/*-----*/
/* increment our key each time we see a new store object */
/* */
/* OBS P P1 P2 P3 P4 V Value */
/* 1 1 stores 0 */
/*-----*/
if v=0 and p=1 and p1="stores" then key+1;

/*-----*/
```

```

/* get StoreName, as in observation 2          */
/*                                           */
/* OBS P  P1  P2  P3      P4  V Value  */
/* 2 2  stores Name          1 Bob's Mart */
/*-----*/
if P=2 and p2 = "Name" then StoreName = value;

/*-----*/
/* get Code and write a storeinfo observation */
/* OBS P  P1  P2  P3      P4  V Value */
/* 17 2  stores Code          1 12BMx2 */
/*-----*/
if p=2 and p2 = "Code" then do;
  Code = Value;
  output storeinfo;
end;

/*-----*/
/* get Count, example observations below     */
/*                                           */
/* OBS P  P1  P2  P3      P4  V Value  */
/* 6 4  stores sales Hot_Dogs count 1 39  */
/* 9 4  stores sales Salami   count 1 20  */
/* 12 4 stores sales Canteloupes count 1 26 */
/*-----*/
if P=4 and p4 = "count" then Count = input(value, 5.);

/*-----*/
/* get Price, and from that same observation, p3 is the Item. And each time */
/* we see price, we want to output an observation.                          */
/*                                           */
/* OBS P  P1  P2  P3      P4  V Value  */
/* 7 4  stores sales Hot_Dogs price 1 1.09 */
/* 10 4 stores sales Salami   price 1 5.99 */
/* 13 4 stores sales Canteloupes price 1 1.39 */
/* 16 4 stores sales Mustard   price 1 2.19 */
/*-----*/
if P=4 and p4 = "price" then do;
  Item = p3;
  Price = input(value, 5.);
  output iteminfo;
end;

run;

```

2つのデータセットの内容は次のとおりです。

proc print data=storeinfo; run; OBS StoreName Code Key 1 Bob's Mart														
12BMx2	12	Grab 'n' Git	10GNx9	23	Larry's Quick Shoppe	17LQx2	3	proc print						
data=iteminfo; run; OBS Item Count Key Price 1 Hot_Dogs 39 1 1.09 2														
Salami	20	1	5.99	3	Canteloupes	26	1	1.39	4	Mustard	6	1	2.19	5
Hot_Dogs	18	2	1.19	6	Salami	3	2	7.99	7	Mustard	6	2	2.19	8
Beer	20	2	8.99	9	Hot_Dogs	39	3	1.09	10	Salami	20	3	5.99	11
Mustard	6	3	2.19	12	Beer	7	3	8.99	13	Wine	15	3	12.99	



次のコードにより 2 つのデータセットをマージします。

```
data a;
  merge storeinfo iteminfo;
  by key;
run;
```

マージされるデータセットの内容は次のとおりです。

```
proc print data=a; var Storename Code Item Count Price; run; OBS StoreName Code
Item Count Price 1 Bob's Mart 12BMx2 Hot_Dogs 39 1.09 2 Bob's Mart
12BMx2 Salami 20 5.99 3 Bob's Mart 12BMx2 Canteloupes 26 1.39 4 Bob's
Mart 12BMx2 Mustard 6 2.19 5 Grab 'n' Git 10GNx9 Hot_Dogs 18 1.19
6 Grab 'n' Git 10GNx9 Salami 3 7.99 7 Grab 'n' Git 10GNx9 Mustard
6 2.19 8 Grab 'n' Git 10GNx9 Beer 20 8.99 9 Larry's Quick Shoppe 17LQx2
Hot_Dogs 39 1.09 10 Larry's Quick Shoppe 17LQx2 Salami 20 5.99 11 Larry's
Quick Shoppe 17LQx2 Mustard 6 2.19 12 Larry's Quick Shoppe 17LQx2 Beer
7 8.99 13 Larry's Quick Shoppe 17LQx2 Wine 15 12.99
```

## JSON Pretty Print DATA ステップ関数の使用法

JSON Pretty Print DATA ステップ関数により、入力 JSON の読み取り可能なコピーが作成されます。次に構文を示します。

```
data _null_;
  rc = jsonpp("input file","output file");
run;
```

### *input file*

入力 JSON です。ここには物理的な名前またはファイル参照名を使用できません。

### *output file*

出力 JSON です。ここには物理的な名前またはファイル参照名を使用できません。出力ファイルが'LOG'の場合、出力は SAS ログに書き込まれます。

次の例では、ディスクファイル test.json が読み取られ、ディスクファイル test.json.pp が作成されます。

```
data _null_;
  rc = jsonpp('test.json', 'test.json.pp');
run;

/* read json from a url fileref and pretty print to disk fileref */

filename in url "http://example.com/~username/snip.json";
filename out 'pp.txt';
data _null_;
  rc = jsonpp('in','out');
run;
```

## 例

### 例 1: LIBNAME ステートメントを使用した JSON マップの作成と再利用

この例では、JSON データを my.json という名前のファイルから読み取り、ファイル my.map 内に JSON マップを作成します。マップが存在しない場合は、エンジンにより作成されます。存在する場合、エンジンは既存のマップを使用しません。

```
filename in 'my.json';
```

```
filename map 'my.map';
libname in json map=map automap=reuse;
```

この例では物理的な JSON ドキュメントの場所を使用して JSON ドキュメントにアクセスする方法を示します。

```
filename map 'my.map';
libname in json 'my.json' map=map automap=reuse;
```

### 例 2: URL アクセス方式ファイル参照名からの JSON の使用法

この例では URL ファイル参照名から JSON にアクセスする方法を示します。

```
filename in url "http://example.com/~username/snip.json" debug;
filename map 'snap.map';
libname in json map=map automap=replace;
proc contents data=in._all_ run;
```

RESTful API を 2 回目に呼び出すには、LIBNAME ライブラリ参照名を再割り当てする必要があります。

```
libname in json ...;
```

---

## LIBNAME ステートメント, WebDAV サーバーアクセス

SAS ライブラリにライブラリ参照名を関連付け、WebDAV (Web-based Distributed Authoring And Versioning)サーバーにアクセスできるようにします。

- 該当要素:** 任意の場所
- カテゴリ:** データアクセス
- 制限事項:** このステートメントは CAS Server ではサポートされていません。
- 

### 構文

```
LIBNAME libref <engine> 'SAS-library' <options> WEBDAV USER="user-ID"
PASSWORD="user-password" WEBDAV options;
```

```
LIBNAME libref CLEAR | _ALL_ CLEAR;
```

```
LIBNAME libref LIST | _ALL_ LIST;
```

### 引数

#### *libref*

SAS ファイルを格納する集約記憶域のショートカット名を指定します。

**ヒント** ライブラリ参照名と SAS ライブラリの関連付けは、SAS セッション終了まで維持されるか、または他の LIBNAME ステートメントで関連付けの変更や関連付けの取り消しを実行するまで維持されます。

#### 'SAS-library'

WebDAV サーバー上の URL の場所(パス)を指定します。URL には、HTTP または HTTPS 通信プロトコルのどちらかを指定します。 .

**制限事項** LIBNAME ステートメントに対して WebDAV 拡張を使用する場合、データライブラリは 1 つのみサポートされます。

---

**要件** HTTPS 通信プロトコルを使用する場合、保護されたネットワーク通信を可能にする TLS (Transport Layer Security) プロトコルを使用する必要があります。詳細については、*Encryption in SAS* を参照してください。

### **engine**

有効な SAS Engine の名前を指定します。

**制限事項** WebDAV オプションを使用する場合、REMOTE Engine はサポートされません。

**参照項目:** 有効なエンジンのリストについては、各動作環境向けの SAS ドキュメントを参照してください。

### **CLEAR**

現在割り当てられている 1 つまたは複数のライブラリ参照名の関連付けを取り消します。WebDAV サーバーを使用するライブラリ参照名の関連付けを取り消すと、ローカルに保存されているキャッシュファイルも削除されます。

**ヒント** 1 つのライブラリ参照名の関連付けを取り消すには、*libref* を指定します。現在割り当てられているライブラリ参照名の関連付けをすべて取り消すには、*\_ALL\_* を指定します。

### **LIST**

1 つまたは複数の SAS ライブラリの属性を SAS ログに書き込みます。

**ヒント** 1 つの SAS ライブラリの属性をリストするには、*libref* を指定します。  
**ト** 現在のセッションにあるライブラリ参照名を含むすべての SAS ライブラリの属性をリストするには、*\_ALL\_* を指定します。

### **\_ALL\_**

現在割り当てられているすべてのライブラリ参照名に対して、CLEAR 引数または LIST 引数を適用するように指定します。

## **LIBNAME オプション**

LIBNAME ステートメントの有効なオプションについては、“[LIBNAME ステートメント](#)” (225 ページ) を参照してください。

## **WebDAV 固有のオプション**

### **WEBDAV**

ライブラリ参照名が WebDAV サーバーにアクセスすることを指定します。

### **USER="user-ID"**

WebDAV サーバーへのアクセスに使用するユーザー名を指定します。ユーザー ID では大文字と小文字が区別されます。また、一重引用符か二重引用符で囲んで指定する必要があります。

**別名** UID

**ヒント** PROMPT を指定して USER= を指定しない場合、ID とパスワードの入力を求めるプロンプトがユーザーに表示されます。

### **PASSWORD="user-password"**

WebDAV サーバーにアクセスするユーザーのパスワードを指定します。パスワードでは大文字と小文字が区別されます。また、一重引用符か二重引用符で囲んで指定する必要があります。

別名 PWD=、PW=、PASS=

ヒント PASSWORD=オプションのかわりに、PROMPT オプションを指定できます。

### PROMPT

必要に応じて、ユーザーのログインパスワードの入力を求めるプロンプトを表示するように指定します。

操作 PROMPT を指定して USER=を指定しない場合、ID とパスワードの入力を求めるプロンプトがユーザーに表示されます。

ヒント PROMPT オプションを指定する場合、PASSWORD=オプションを指定する必要はありません。

### PROXY=*url*

プロキシサーバーの URL(Uniform Resource Locator)を次のどちらかの形式で指定します。

- "http://hostname"
- "http://hostname:port"

### LOCALCACHE="*directory name*"

サーバーファイルのローカルコピーを格納するための一時ディレクトリを作成するディレクトリを指定します。ライブラリ参照名ごとに重複しないサブディレクトリが用意されます。ディレクトリを指定しない場合、SAS WORK ディレクトリにサブディレクトリが作成されます。一時ファイルは SAS プログラムが終了するときに削除されます。

デフォルト SAS WORK ディレクトリ

### LOCKDURATION=*n*

WebDAV ライブラリ参照名から書き込むファイルをロックする時間(分)を指定します。一時ファイルは SAS プログラムが正常に終了するときに削除されます。SAS プログラムに問題が発生すると、指定した時間が経過した後にロックが解除されます。

デフォルト 30

## 詳細

### WebDAV サーバーの使用時に機能が異なるデータセットオプション

WebDAV サーバーの使用時に機能が異なるデータセットオプションを次の表に示します。他のすべてのデータセットオプションは、*SAS Viya Data Set Options: Reference* で説明されているように機能します。

表 2.8 WebDAV サーバーの使用時のデータセットオプションの機能

データセットオプション	WebDAV ストレージでの機能
CNTLLEV=	<p><i>LIB</i> は、データをローカルキャッシュに書き込む前に、ライブラリにあるすべてのデータセットをロックします。DATA ステップが完了するとすべてのメンバのロックが解除され、データセットが WebDAV サーバーに書き込まれます。</p> <p><i>MEM</i> は、ローカルキャッシュにデータを書き込む前にメンバをロックします。DATA ステップが完了するとメンバのロックが解除され、データが WebDAV サーバーに書き込まれます。</p> <p><i>REC</i> はサポートされません。WebDAV では、データセット全体に対する更新のみが許可されます。</p>
FILECLOSE	VxTAPE エンジンにはサポートされていません。よって、このオプションは無視されます。
GENMAX=	WebDAV サーバーには保持する変更履歴の最大数を指定できないため、この機能はサポートされません。
GENNUM=	WebDAV では、この機能はサポートされていません。
IDXNAME=	インデックスが存在する場合、使用するインデックスを指定できます。
INDEX=	インデックスをローカルキャッシュに作成し、WebDAV サーバーに保存することができます。
TOBSNO=	リモートエンジンはサポートされていません。よって、このオプションは無視されます。

### WebDAV でのファイル処理

WebDAV サーバーにアクセスすると、ファイルは WebDAV サーバーから処理を実行するローカルディスクストレージに取得されます。更新が完了すると、ファイルは格納先の WebDAV サーバーに送信されます。このファイルは、送信後にローカルディスクから削除されます。

### WebDAV ライブラリに対する複数のライブラリ参照名

WebDAV サーバー上のファイルにライブラリ参照名を割り当てると、パス(URL の場所)、ユーザー ID、パスワードがライブラリ参照名に関連付けられます。最初のライブラリ参照名の割り当て後に、同じライブラリに対して別のライブラリ参照名を割り当てる場合、このユーザー ID とパスワードが検証されます。

注: 後から同じライブラリに対して別のライブラリ参照名を割り当てる際に、別のユーザー ID または同じパスワード、あるいはその両方を使用すると、通常は発生しないロックエラーが起こる場合があります。

### WebDAV サーバー上でのファイルのロック

ローカルライブラリでは、読み取り中に他のユーザーによってファイルが変更されるのを防ぐため、ファイルを開くときにファイルがロックされます。WebDAV でロックするには、ライブラリへの書き込みアクセス権が必要です。読み取りロックの概念はありません。また、WebDAV サーバーはいつでも、停止、起動、オフラインを実行することができます。そのため、ファイルが他のユーザーによってすでにロックされている場合にのみ、WebDAV サーバー上のファイルに対するロックリクエストを実行します。

### 例: WebDAV ディレクトリにライブラリ参照名を関連付ける

次の例では、ライブラリ参照名 **davdata** を WebDAV サーバー **www.webserver.com** 上の WebDAV ディレクトリ **/users/mydir/datadir** に関連付けます。

```
libname davdata v9 "https://www.webserver.com/users/mydir/datadir"
    webdav user="mydir" pw="12345";
```

### 関連項目:

#### ステートメント:

- [“LIBNAME ステートメント” \(225 ページ\)](#)

---

## LINK ステートメント

指定したステートメントラベルにプログラム実行移動します。RETURN ステートメントが後に続く場合、LINK ステートメントの直後のステートメントに実行を戻します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御
種類:	実行

---

### 構文

**LINK** *label*;

### 引数

#### *label*

LINK の移動先となるステートメントラベルを指定します。*label* 引数は必ず指定する必要があります。

### 詳細

LINK ステートメントは、LINK ステートメントに指定したステートメントラベルに SAS を移動します。移動先から RETURN ステートメントまでの間にあるステートメントの実行を続けます。RETURN ステートメントは、LINK ステートメントの後に続くステートメントにプログラム制御を移動します。

LINK ステートメントと移動先は同じ DATA ステップ内になければなりません。LINK ステートメントでは、移動先はステートメントラベルで指定します。

LINK ステートメントでは、別の LINK ステートメントを含むステートメントグループに移動することができます。この配置方法はネストと呼ばれます。無限ループが発生しないように、ネストする LINK ステートメントのデフォルト数が SAS によって設定されています。RETURN ステートメントを間にはさまずに、最大 10 個までの LINK ステートメントをネストできます。複数の LINK ステートメントが実行された場合、RETURN ステートメントは最後に実行した LINK ステートメントの次のステートメントに SAS を戻します。ネストする LINK ステートメントの数を増やすには、DATA ステートメントで /STACK オプションを使用します。

## 比較

LINK ステートメントと GO TO ステートメントでは、後続く RETURN ステートメントのアクションが異なります。LINK ステートメントの後に RETURN ステートメントを指定すると、LINK ステートメントの直後のステートメントに実行を戻します。GO TO ステートメントの後に RETURN ステートメントを指定し、GO TO ステートメントの前に LINK ステートメントがない場合は、DATA ステップの先頭に戻ります。このとき、GO TO ステートメントの前に LINK ステートメントがある場合は、LINK ステートメントの直後の最初のステートメントから実行を続けます。通常、LINK ステートメントは RETURN ステートメントを明示的に指定して使用しますが、GO TO ステートメントは RETURN ステートメントを指定せずに使用します。

同じプログラム内の複数のポイントでステートメントグループを実行する場合、LINK ステートメントを使用するとコーディングが簡単になり、プログラムロジックも理解しやすくなります。プログラム内でステートメントグループを 1 つのポイントでのみ実行する場合、LINK-RETURN ロジックよりも DO グループロジックを使用するほうが簡単です。

## 例: プログラム実行の移動

この例では、変数 TYPE の値が **aluv** になる場合、LINK ステートメントはラベル CALCU に割り当てたステートメントにプログラム実行を移動します。RETURN ステートメントが検出されるまでこのプログラムが実行されます。RETURN ステートメントが検出されると、プログラム実行は LINK の後に続く最初のステートメントに移動されます。割り当てステートメントを実行し、オブザベーションを書き込みます。その後、DATA ステップの先頭に戻って、次のレコードを読み込みます。TYPE の値が **aluv** ではない場合、割り当てステートメントを実行し、オブザベーションを書き込みます。その後、DATA ステップの先頭に戻ります。

```
data hydro;
  input type $ depth station $;
  /* link to label calcu: */
  if type ='aluv' then link calcu;
  date=today();
  /* return to top of step */
  return;
  calcu: if station='site_1'
    then elevatn=6650-depth;
  else if station='site_2'
    then elevatn=5500-depth;
  /* return to date=today(); */
  return;
  datalines;
aluv 523 site_1
uppa 234 site_2
aluv 666 site_2
```

```
...more data lines...
```

```
;
```

## 関連項目:

### ステートメント:

- [“DATA ステートメント” \(47 ページ\)](#)
- [“DO ステートメント” \(62 ページ\)](#)
- [“GO TO ステートメント” \(132 ページ\)](#)
- [“ラベル: ステートメント” \(219 ページ\)](#)
- [“RETURN ステートメント” \(365 ページ\)](#)

---

## LIST ステートメント

現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。

**該当要素:** DATA ステップ

**カテゴリ:** アクション  
CAS

**種類:** 実行

---

## 構文

```
LIST;
```

## 引数なし

LIST ステートメントを使用すると、現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。

## 詳細

LIST ステートメントは、INPUT ステートメントで読み込まれるデータのみを処理します。SET、MERGE、MODIFY、UPDATE の各ステートメントで読み込まれるデータは処理されません。

SAS ログでは、列の位置を示すルーラーが表示してから、レコードを表示します。

可変長レコード(RECFM=V)の場合、入力行の最後にレコード長を書き込みます。固定長レコード(RECFM=F)の長さは、読み込むデータ量がレコード長(LRECL)と一致している場合には書き込みません。

## 比較

次の表で、LIST ステートメントと PUT ステートメントを比較します。



アクション	LIST ステートメント	PUT ステートメント
書き込みのタイミング	DATA ステップの繰り返しの最後に実行	すぐ実行
書き込み内容	入力した内容と同一の入力データレコード	指定した変数またはリテラル
書き込み先	SAS ログのみ	SAS ログ、SAS 出力先、外部ファイル
処理可能なステートメント	INPUT ステートメントのみ	すべてのデータ読み込みステートメント
16 進数の値の処理	書き込みできない文字が検出された場合は自動的に値を 16 進数で出力する	16 進数の出力形式が指定された場合にのみ文字を 16 進数で表示する

## 例

### 例 1: 欠損データを含むレコードをリストする

この例では、LIST ステートメントを使用して、欠損データが含まれる入力レコードを SAS ログに書き込みます。INPUT ステートメントの行ポインタコントロールが #3 と指定されているため、1 つのオブザベーションを作成するのに 3 つのレコードを読み込みます。そのため、W2AMT の値が欠損しているたびに、現在の入力レコードが 3 行にわたって SAS ログに書き込まれます。

```
data employee;
  input ssn 1-9 #3 w2amt 1-6;
  if w2amt=. then list;
  datalines;
23456789
JAMES SMITH
356.79
345671234
Jeffrey Thomas
:
;
```

#### アウトプット 2.17 欠損データを表示するログ

```
RULE:-----1-----2-----3-----4-----5----- 9 345671234 10
Jeffrey Thomas 11 .
```

番号 9、10、と 11 は SAS ログ内の行番号を示しています。

### 例 2: 可変長レコードのレコード長を表示する

この例では、可変長 ID 番号を含むの外部ファイルを入力として使用します。RECFM=V オプションは INFILE ステートメントで指定され、LIST ステートメントは SAS ログにレコードを書き出します。この例の RECFM=V オプションが示すように、ファイルが可変長のレコードを持つ場合、SAS ログにリストされる各レコードの最後にレコード長が書き込まれます。

```
data employee;
  infile 'your-external-file' recfm=v;
  input id $;
  list;
run;
```

#### アウトプット 2.18 可変長レコードどレコード長を表示するログ

```
RULE: -----1-----2-----3-----4-----5--- 1 23456789 8 2
123456789 9 3 5555555555 10 4 345671234 9 5 2345678910 10 6 2345678 7
```

#### 関連項目:

#### ステートメント:

- [“PUT ステートメント” \(318 ページ\)](#)

---

## %LIST ステートメント

現在のセッション中に入力された行を表示します。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

### 構文

```
%LIST<n <:m | - m> >;
```

#### 引数なし

対話型ラインモードの処理で引数を指定せずに%LIST ステートメントを使用すると、これまでに入力したすべてのプログラム行が表示されます。

#### 引数

*n*  
*n* 番目の行を表示します。

*n-m*  
*n* 番目から *m* 番目までの行を表示します。

別名 *n:m*

### 詳細

#### 指定可能な位置および指定する必要がある場合

%LIST ステートメントは SAS ジョブ内のどこにでも指定することができますが、DATALINES ステートメントと終わりを示すセミコロン(;)の間または DATALINES4 ステートメントと終わりを示す複数のセミコロン(;;;)の間には使用できません。このステートメントは、対話型ラインモードで SAS プログラムコ

ードをモニタ上に表示する場合に便利です。また、%INCLUDE ステートメントの使用時に取得する行を決定する場合にも使用できます。

## 対話

### 注意:

SPOOL システムオプションは、SAS ステートメントを保存するかどうかをすべての実行モードで制御します。対話型ラインモードで SPOOL システムオプションが有効に設定されている場合、すべての SAS ステートメントとデータ行はサブミット時に自動的に保存されます。%%LIST ステートメントを使用すると、これらを表示することができます。NOSPOOL オプションが有効に設定されている場合、%LIST ステートメントを使用しても以前に入力した行は表示されません。

## 例: 現在のセッション中に入力された行を表示する

次の%LIST ステートメントでは 10 番目から 20 番目の行を表示します。

```
%list 10-20;
```

## 関連項目:

### ステートメント:

- ["%INCLUDE ステートメント" \(138 ページ\)](#)

### システムオプション:

- ["SPOOL System Option" \(SAS Viya System Options: Reference\)](#)

---

## LOCK ステートメント

既存の SAS ファイルに対する排他的ロックの取得、リスト出力、または解除を行います。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

現在、他の SAS セッションがアクセスしている SAS ファイルをロックすることはできません(排他的ロックされている、そのファイルが開かれているためロックできません)。

LOCK ステートメントの構文は、シングルユーザー環境でも、クライアント/サーバー環境でも同じです。ただし、LOCK ステートメントの機能によっては、クライアント/サーバー環境にのみ適用される場合があります。

---

## 構文

```
LOCK libref<.member-name<.member-type | .entry-name.entry-type>>  
<LIST | QUERY | SHOW | CLEAR | NOMSG>;
```

## 引数

### **libref**

SAS ライブラリに関連付けられた名前を指定します。libref (ライブラリ参照名)には、有効な SAS 名を指定する必要があります。libref が WORK の場合、それを指定する必要があります。

ヒント シングルユーザー環境では、LOCK ステートメントを発行してライブラリを排他的にロックする必要はありません。

### **member-name**

ライブラリ参照名に関連付けられた SAS ライブラリのメンバを示す有効な SAS 名を指定します。

制限事項 ロックを要求する前に、SAS ファイルを作成する必要があります。

### **member-type**

ロック対象とする SAS ファイルの種類を指定します。たとえば、有効な値には DATA、VIEW、CATALOG、MDDDB などがあります。デフォルトでは DATA に設定されます。

### **entry-name**

ロック対象のカタログエントリの名前を指定します。

ヒ シングルユーザー環境で LOCK ステートメントを発行して個別のカ  
ン ログエントリをロックした場合、カタログ全体がロックされます。通  
ト 常、カタログエントリを排他的にロックするために LOCK ステート  
メントを発行するようなことはしません。

### **entry-type**

ロック対象のカタログエントリの種類を指定します。

ヒ シングルユーザー環境で LOCK ステートメントを発行して個別のカ  
ン ログエントリをロックした場合、カタログ全体がロックされます。通  
ト 常、カタログエントリを排他的にロックするために LOCK ステート  
メントを発行するようなことはしません。

## **LIST | QUERY | SHOW**

特定の SAS ファイルに排他的ロックを設定したかどうかを SAS ログに書き込みます。

ヒント このオプションは、クライアント/サーバー環境ではさらに詳しい情報を提供します。

## **CLEAR**

SAS セッションで LOCK ステートメントを使用して取得した、特定の SAS ファイルに対するロックを解除します。

## **NOMSG**

警告やエラーメッセージが SAS ログに出力されないように指定します。NOMSG では、ロックが成功した、またはロックが解除された、と告げるノートを非表示にはしません。

操 警告やエラーメッセージを非表示にするには、LOCK ステートメントを  
作 実行する度に NOMSG を指定してください。

SAS マクロ変数 SYSLOCKRC の戻り値は、NOMSG が指定されていても影響されません。

- ヒ  
ン  
ト
- NOMSG は、ロックが利用可能になるまで LOCK ステートメントをコードループでサブミットし続ける時に、排他的ロックが利用できないというエラーメッセージは毎回 SAS ログに表示したいという場合に有効です。

## 詳細

### 概要情報

LOCK ステートメントでは、すでに存在する SAS ファイルに対する排他的ロックの取得、リスト出力、または解除を実行できます。排他的ロックでは、ロックが解除されるまでは、現在の SAS セッションでその他の操作によってファイルの読み込み、書き込み、またはロックを行うことはできなくなります。さらに、排他的ロックでは、ファイルが開いている場合、ロックによって確実に別の SAS セッションがそのファイルにアクセスできないようにします。

LOCK ステートメントの主な用途は、SAS ステートメント境界を越えて SAS ファイルの排他的制御を保持することです。ファイルの排他的制御を失わずに、ファイルに対して複数の操作を次々に実行することが望ましい場合もあります。ただし、DATA または PROC ステップが実行を終了し、制御が次の操作に移ると、ファイルは閉じられ、同じデータ記憶域の場所へのアクセス権を持つ別の SAS セッションによる処理が可能になります。

排他的ロックを解除するには、CLEAR オプションを使用します。また、データセットに対する排他的ロックは、DATASETS プロシジャの DELETE ステートメントを使用してそのデータセットを削除すると解除されます。

### LOCK ステートメントのリターンコード

SAS マクロ変数 SYSLCKRC には、LOCK ステートメントからのリターンコードが含まれます。次のアクションは、SYSLCKRC に 0 以外の値を挿入します。

- ファイルをロックしようとしたが、ロックを取得できない(たとえば、ファイルが使用されている、またはファイルが他の SAS セッションでロックされているなど)
- LOCK ステートメントに LIST オプションを指定してロックの一覧を表示する
- LOCK ステートメントに CLEAR オプションを指定してユーザーが取得していないロックを解除する

SYSLCKRC SAS マクロ変数の詳細については、[SAS Viya マクロ言語リファレンス](#) を参照してください。

## 比較

CNTLLEV=データセットオプションでは、SAS データセットへの共有更新アクセスを拒否するレベルを指定できます。

### 例: SAS ファイルをロックする

次の SAS プログラムでは、SAS データセットのロックプロセスを説明します。LOCK ステートメントを使用すると、ファイルに対して排他的アクセスが取得されるため、マルチステッププログラムに保護が提供されます。

```
libname mydata 'SAS-library';
lock mydata.census; 1
data mydata.census; 2
```

```

    modify mydata.census;
    (statements to remove obsolete observations)
run; 3
proc sort force data=mydata.census; 4
    by CrimeRate;
run;
proc datasets library=mydata; 5
    modify census;
    index create CrimeRate;
quit;
lock mydata.census clear; 6

```

- 1 SAS データセット MyData.Census に対して排他的アクセスを取得します。
- 2 MyData.Census を開き、オブザベーションを削除します。更新中は、現在の SAS セッションにおける他の操作や、他の SAS セッションで、このファイルにアクセスすることはできません。
- 3 DATA ステップの最後にファイルが閉じられます。現在の SAS セッションにおいて他の操作でこのファイルにアクセスすることはできません。ただし、このファイルが再度開くまでは、別の SAS セッションでアクセスできます。
- 4 MyData.Census を開き、ファイルを並べ替えます。並べ替え中は、現在の SAS セッションにおける他の操作や、他の SAS セッションで、このファイルにアクセスすることはできません。プロシジャの最後に、ファイルが閉じられます。これは、現在の SAS セッションにおいて他の操作でこのファイルにアクセスすることはできませんが、別の SAS セッションでのアクセスはできることを意味します。
- 5 MyData.Census を開き、インデックスを再構築します。現在の SAS セッションにおける他の操作や、他の SAS セッションで、このファイルにアクセスすることはできません。プロシジャの最後にファイルが閉じられます。
- 6 MyData.Census に対する排他的ロックを解除します。

## 関連項目:

### データセットオプション:

- [“CNTLLEV= Data Set Option” \(SAS Viya Data Set Options: Reference\)](#)

---

## LOCKDOWN ステートメント

サーバープロセスの内部からホスト動作環境へのアクセスを制限することにより、ワークスペースまたは SAS/CONNECT サーバーを保護します。

**該当要素:** AUTOEXEC ファイルまたは INITSTMT=システムオプション

**カテゴリ:** 制御

**種類:** 実行

## 構文

**LOCKDOWN ENABLE\_AMS**=*access-method-1* <...*access-method-n*>;

**LOCKDOWN FILE**=*fileref*;

**LOCKDOWN PATH**='pathname-1' <...'pathname-n'>;

**LOCKDOWN LIST**;

## 引数

**ENABLE\_AMS**=*access-method-1* <...*access-method-n*>;

デフォルトでは、SAS がロックダウン状態にある場合、次のアクセス方式は使用できません。

- URL
- HTTP

ENABLE\_AMS を使うことで、管理者はロックダウン状態ではデフォルトで無効化されているアクセス方式やプロシジャを再有効化できます。

アクセス方式の名前では大文字/小文字が区別されません。重複する名前は無視されます。名前と名前をスペースで区切り、引用符を使用しないでください。

URL/HTTP は別名が付けられた名前のペアであり、したがって一方を再有効化すると、他方も自動的に再有効化されます。

ENABLE\_AMS=は、AUTOEXEC ファイルまたは INITSTMT=オプションにのみ含めることができます。

**FILE**=*fileref*

ロックダウンパスリストに追加する有効なディレクトリおよびファイルのリストを含むファイルを指定します。リスト内の任意の有効なディレクトリのサブディレクトリが、SAS により自動的にロックダウンパスリストに追加されます。

ロックダウンパスリストは、ロックダウン状態にあるときにどのファイルとディレクトリに SAS セッションがアクセスできるのかを指定するリストです。(このリストはホワイトリストと呼ばれています。)

ロックダウンファイルに複数の行を含めることができます。各行によりパス文字列を指定します。各行のパス文字列として、1つのパス名またはパス名指定の連結を使用できます。行にパス名が1つのみある場合、そのパス名にスペースまたはホストでサポートされるパス名文字を含めることができます。このパス名を引用符で囲む必要はありません。

行に連結指定がある場合は、連結したパス指定のグループ全体をカッコで囲みます。それぞれのパス名を引用符で囲みます。パス名指定の間をスペース文字で区切ります。パス文字列内で改行文字を使用しないでください。詳細については、“[PATH='pathname'](#)” (276 ページ)を参照してください。

ロックダウンテキストファイル内のパスリストに DBCS 文字がある場合は、ENCODING オプションを FILENAME ステートメントに追加します。この追加により、ロックダウンパスリストが適切に SAS セッションエンコーディングにトランスコードされることが保証されます。たとえば、DBCS 文字を使用したパスリストを含むテキストファイル lockdown-text-file.txt がある場合には、サーバーの AUTOEXEC ファイル appserver\_autoexec.sas に次のコードを追加します。

```
filename myfile "c:\lockdown-text-file.txt" encoding='euc-cn';
lockdown file=myfile;
lockdown list;
```



**PATH='pathname'**

ロックダウンパスリストに追加する 1 つ以上のパスを指定します。パス名は引用符で囲みます(SAS マクロ変数との競合を回避するには、単一引用符をお勧めします)。複数のパス名の間をスペース文字で区切ります。

ロックダウンパスリストは、ロックダウン状態にあるときにどのファイルとディレクトリに SAS セッションがアクセスできるのかを指定するリストです。(このリストはホワイトリストと呼ばれています。)リスト内の任意の有効なディレクトリのサブディレクトリが、SAS により自動的にロックダウンパスリストに追加されます。

パス名は相対でも絶対でも指定できます。パス名に、オペレーティングシステムの環境変数、SAS 環境変数(たとえば !SASROOT)、現在の作業ディレクトリ(.)および一般にパス名で使用できるその他のホスト固有の文字置換を含めることもできます。たとえば UNIX のパスでは~が受け入れられます。

注: SAS 環境変数はパス名の先頭에만指定できます。

**LIST**

ロックダウンパスリスト内の現在の有効なパスが SAS ログに出力されます。

このリストは最適化されたパスリストです。サブディレクトリおよび重複するパス名は表示されません。

**詳細**

LOCKDOWN ステートメントを使用すると、サーバー内ではまたはバッチ処理モードで実行される SAS セッションのための、ローカルファイルおよび特定 SAS 機能へのアクセスを制限できます。

LOCKDOWN ステートメントを使用するには、影響を受けるサーバーの sasv9\_usermods.cfg で LOCKDOWN システムオプションを指定する必要があります。詳細については、“[LOCKDOWN System Option](#)” (*SAS Viya System Options: Reference*)を参照してください。

ロックダウン状態にある SAS サーバーにより、ロックダウンパスリストを通じてホストファイルシステムへのすべてのアクセスが検証されます。

ロックダウンパスリストには、特定のサーバーにアクセスできるすべてのパスが含まれます。(このリストはホワイトリストと呼ばれています。)ロックダウンパスリスト内のパスが存在するかどうかは SAS により確認されません。ロックダウンホワイトリストに含まれるディレクトリに対するオペレーティングシステムの許可はそれでも効果を持ちます。

あるパスがホワイトリスト内で宣言されていても、そのパスの任意ファイルを任意のユーザーが読み取れることにはなりません。LOCKDOWN ステートメント以前と同様に、物理的なファイルやディレクトリに対するホストの許可は、常にホワイトリストに優先します。

ホワイトリストの実装方法に関する提案については、“[例 2: ホワイトリスト外のパスの位置付けによるホワイトリストの非表示化](#)” (278 ページ)を参照してください。

ロックダウンパスリストは SAS の初期化中に作成され、ロックダウンの時点で完成します。

ロックダウンパスリスト内には、デフォルトのロックダウンディレクトリ(SAS 設定ファイルからのパス)およびユーザーのディレクトリとファイル(サーバーの autoexec ファイル内の LOCKDOWN ステートメントを使用して追加)という 2 種類のパスがあります。

ホワイトリストを変更(新しいストアドプロセスリポジトリの定義を含む)しても現在実行されているサーバーに影響しません。変更を行った後で、影響を受ける



プールワークスペースサーバーまたはストアプロセスサーバー上で現在実行されているプロセスを休止または停止してください。変更点は、次回に SAS によりサーバーセッションが開始される時に有効になります。

詳細については、“[Lock Down Workspace Servers](#)” (*SAS Viya Administration: Workspace Server and Object Spawner*)および“[Lock Down the SAS/CONNECT Server](#)” (*SAS Viya Administration: SAS/CONNECT Server and Spawner*)を参照してください。

SAS により、SAS 設定ファイルおよび SAS サーバーメタデータ定義に含まれる次の定義済みのパスがデフォルトで追加されます。

- すべてのストアプロセスのパス, ソースコードリポジトリ
- SASROOT パス
- 現在の作業ディレクトリ(ユーザーのホームディレクトリではない)
- SASAUTOS オプションパスまたは環境変数パス
- UTILLOC オプションパス
- LOGPARM 定義の LOG ファイルパス
- MAPS オプションパス
- JAVA\_HOME/lib/fonts

注: あらかじめ割り当てられたライブラリパスは、自動的にロックダウンリストに追加されません。ユーザーはメタデータまたは autoexec により作成されるライブラリ参照名を通じて、あらかじめ割り当てられたライブラリにアクセスできます。

サーバーの AUTOEXEC ファイルまたは INITSTMT ステートメント内で LOCKDOWN ステートメントを発行すると、他の有効なパスを追加できます。一般に、ステートメントはサーバーの autoexec\_usermods.sas ファイル内で追加されます。

SAS サーバーのタイプごとに、異なるロックダウンパスリストが必要になることがあります。たとえば、ワークスペースサーバーが個別ユーザーのホームディレクトリにのみアクセスできることが必要になる場合があります。あるいは、ストアプロセスサーバーが、より大きな特権を必要とする場所にアクセスすることが必要になる場合があります。もちろん、メタデータや autoexec 内であらかじめ割り当てられたライブラリにより作成されるライブラリ参照名もユーザーは利用できます。

## 例

### 例 1: ワークスペースまたは SAS/CONNECT サーバー経由のユーザーのファイルへのアクセスの有効化

クライアントユーザーの個人アカウントの下で起動されたワークスペースまたは SAS/CONNECT サーバーの場合、管理者はそのユーザーの個人ファイルへのアクセスを有効化するものの他のユーザーへのアクセスを禁止できます。別個のサーバー定義を作成するか特別なロジックを実装して、サーバーの autoexec でカスタマイズした LOCKDOWN ステートメントをサブミットする必要はありません。そのかわりに、管理者はサーバー起動時に使用されたアカウントの所有者であるユーザーのホームディレクトリを参照する特別なロックダウンパスのフォームを指定できます。このフォームは、サーバーを稼働させているホストプラットフォームに固有です。Linux 用の宣言は `lockdown path='~';` です。

**例 2: ホワイトリスト外のパスの位置付けによるホワイトリストの非表示化**

SAS 管理者はホワイトリスト上にないパスにホワイトリストファイルを位置付けることにより、ホワイトリストの内容を非表示にできます。

ホワイトリストファイル内で有効なパスをすべて定義します。たとえば、whitelist.txt に次のコードを追加します。

```
/valid/path1
/valid/path2
```

ホワイトリストファイルを指す **LOCKDOWN file**=ステートメントを追加してサーバーの autoexec ファイルを変更します。この例では、**/opt/sas/viya/config/etc/lockdown** はロックダウンホワイトリストでは定義されていません。

```
filename lkdn "/opt/sas/viya/config/etc/lockdown/whitelist.txt";
lockdown file = lkdn;
```

**例 3: ロックダウンモード中の FTP の有効化**

この例では、管理者は SAS が LOCKDOWN モードにある間に、特定のパブリック FTP サイトへのユーザーアクセスを許可します。そのために、管理者は次のグローバルステートメントをサーバーの autoexec に追加します:

```
lockdown enable_ams = FTP;
```

**関連項目:**

- [“Lock Down Workspace Servers” \(SAS Viya Administration: Workspace Server and Object Spawner\)](#)
- [“Lock Down the SAS/CONNECT Server” \(SAS Viya Administration: SAS/CONNECT Server and Spawner\)](#)
- [“Types of Sign-ons” \(SAS/CONNECT for SAS Viya: User’s Guide\)](#)

**システムオプション**

- [“LOCKDOWN System Option” \(SAS Viya System Options: Reference\)](#)

---

**LOSTCARD ステートメント**

1 オブザベーションあたりのレコードが複数あるデータに欠損レコードや無効なレコードが検出された際に、入力データを再同期します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション  
CAS

**種類:** 実行

---

**構文**

**LOSTCARD;**

## 引数なし

LOSTCARD ステートメントを使用すると、現在のグループに欠損レコードが含まれている場合、次のグループからレコードを読み込まないようにします。

## 詳細

### LOSTCARD を使用する必要がある場合

複数のレコードを読み込んで 1 つのオブザベーションを作成する場合、データの最後に到達するまでレコードの欠損は検出されません。データ内に欠損レコードが存在すると、SAS データセットの後続のオブザベーションの値に誤りが生じる可能性があります。LOSTCARD ステートメントを使用すると、現在のグループのレコード数が予測よりも少ない場合、次のグループからデータを読み込まないようにします。

LOSTCARD ステートメントは、1 オブザベーションあたりの入力データのレコード数が固定されている場合や、オブザベーションの各レコードに同じ値の ID 変数を使用されている場合に便利です。通常、LOSTCARD ステートメントは、IF-THEN ステートメントの THEN 句や SELECT グループのステートメントなどの条件付き処理の中で使用されます。

### LOSTCARD ステートメントの実行

LOSTCARD ステートメントを実行すると、次に示す複数のステップが実行されます。

1. LOSTCARD メッセージ、ルーラー、現在のオブザベーションを作成しようとして読み込んだすべてのレコードの 3 つが SAS ログに書き込まれます。
2. 読み込み中のレコードグループにある最初のレコードを破棄し、オブザベーションには書き込まずに、DATA ステップの先頭に処理を戻します。
3. 自動変数\_N\_の値に 1 を追加しません。(通常は、DATA ステップの先頭に戻ったときに自動変数\_N\_の値を 1 ずつ増やしていきます。)
4. グループ内の 2 番目のレコードから読み込みを開始して、オブザベーションを作成します。INPUT ステートメントに指定されている数のレコードを読み込みます。
5. LOSTCARD の IF 条件が真である場合、ステップ 1 から 4 を繰り返します。ログを見やすくするため、指定したレコードグループに対してメッセージとルーラーが出力されるのは 1 度だけです。また、レコードが、次のオブザベーションの作成で使用される場合でも 1 度しか出力されません。
6. LOSTCARD の IF 条件が真でない場合、オブザベーションが作成され、SAS データセットに書き込まれます。

## 例: 入力データの再同期

この例では、条件付きの作成に LOSTCARD ステートメントを使用して、欠損データの特定と入力データの再同期を実行します。

```
data inspect;
  input id 1-3 age 8-9 #2 id2 1-3 loc
        #3 id3 1-3 wt;
  if id ne id2 or id ne id3 then
  do;
    put 'DATA RECORD ERROR: ' id= id2= id3=;
    lostcard;
  end;
```

```

datalines;
301 32
301 61432
301 127
302 61
302 83171
400 46
409 23145
400 197
411 53
411 99551
411 139
;

```

この DATA ステップでは、オブザベーションを書き込む前に入力レコードを 3 つ読み込みます。レコード 1(変数 ID)の ID 番号が 2 番目のレコード(ID2)または 3 番目のレコード(ID3)と一致しない場合、レコードが正しく入力されていないか欠損しています。IF-THEN DO ステートメントでは、ID 番号が無効な場合に PUT ステートメントに指定したメッセージを出力し、LOSTCARD ステートメントを実行するように指定しています。

この例では、2 番目のオブザベーション(ID3=400)の 3 番目のレコードが欠損しています。3 番目のオブザベーションの 2 番目のレコードが誤って入力されています(ID=400 となるのが正しいですが、ID2=409 となっています)。そのため、データセットには ID301 と ID411 の 2 つのオブザベーションが含まれます。ID=302 または ID=400 のオブザベーションはデータセットに含まれません。この DATA ステップを実行すると、PUT ステートメントと LOSTCARD ステートメントにより、次のステートメントがログに書き込まれます。

```

DATA RECORD ERROR: id=302 id2=302 id3=400 NOTE: LOST CARD.RULE:----+----1----+----2----+----3----
+----4----+----5----+---- 14 302 61 15 302 83171 16 400 46 DATA RECORD ERROR: id=302 id2=400 id3=409
NOTE: LOST CARD.17 409 23145 DATA RECORD ERROR: id=400 id2=409 id3=400 NOTE: LOST CARD.18
400 197 DATA RECORD ERROR: id=409 id2=400 id3=411 NOTE: LOST CARD.19 411 53 DATA RECORD
ERROR: id=400 id2=411 id3=411 NOTE: LOST CARD.20 411 99551

```

番号 14、15、16、17、18、19、20 は SAS ログ内の行番号を示しています。

## 関連項目:

### ステートメント:

- [“IF-THEN/ELSE ステートメント” \(137 ページ\)](#)

---

## MERGE ステートメント

複数の SAS データセットにある複数のオブザベーションを 1 つのオブザベーションに結合します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 実行

**注:** MERGE ステートメントを使用して読み込まれた値は PDV で保持されます。読み込まれた変数のデータタイプも保持されます。[“RETAIN ステートメント” \(361 ページ\)](#)を参照してください。

## 構文

```

MERGE SAS-data-set-1 <(data-set-options)>
SAS-data-set-2 <(data-set-options) >
<...SAS-data-set-n<(data-set-options)> >
<END=variable>;

```

## 引数

### SAS-data-set

オブザベーションを読み込む既存の SAS データセットを少なくとも 2 つ指定します。データセットは個別に指定するか、データセットのリストで指定します。また、両方を組み合わせても指定できます。

**ヒント** データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

追加する SAS データセットを指定できます。

**参照項目:** [“MERGE ステートメントでデータセットリストを使用する” \(282 ページ\)](#)

### (data-set-options)

SAS データセット名の後ろに、1 つまたは複数の SAS データセットオプションを丸かっこで囲んで指定します。

**注** データセットオプションは、処理対象のオブザベーションを DATA ステップを読み込むときに実行するアクションを指定します。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 Viya Data Set Options: Reference](#)

**ヒント** データセットリストに適用するデータセットオプションは、データセットリストに存在するすべてのデータセットに適用されます。

### END=variable

作成する一時変数の名前を指定します。この変数の値には終端指示子が格納されません。

**注** この変数は 0 に初期化されますが、MERGE ステートメントが最後のオブザベーションを処理するときに 1 に設定されます。各入力データセットに存在するオブザベーションの数が異なる場合は、MERGE ステートメントによってすべてのデータセットの最後のオブザベーションが処理されるときに、END=変数が 1 に設定されます。

**ヒント** END=変数は、作成される SAS データセットには追加されません。

## 詳細

### 概要

MERGE ステートメントには柔軟性があり、SAS プログラミングでさまざまな方法で使用されます。次のセクションでは、MERGE ステートメントの基本的な使用方法を説明します。応用として、2 つ以上の BY 変数の使用、3 つ以上のデータセットのマージ、少数のオブザベーションを別のデータセットの全オブザベーションへのマージなどがあります。

### MERGE ステートメントでデータセットリストを使用する

MERGE ステートメントでは、データセットのリストを使用できます。データセットリストを使用すると、現在存在するデータセットのグループを簡単に示せます。このデータセットリストには、名前接頭辞リストまたは番号付き範囲リストを指定する必要があります。

**名前接頭辞リスト**は、指定した文字列で始まるすべてのデータセットを示します。たとえば、**merge SALES1;**と指定すると、"SALES1"で始まる SALES1、SALES10、SALES11、SALES12 などのデータセットがすべてマージされます。

**番号付き範囲リスト**では、連番である最後の文字を除き、同じ名前のデータセットが存在する必要があります。番号付き範囲リストでは、開始値と終了値に任意の数値を使用してもかまいません。たとえば、次のリストは同じデータセットを示します。

```
sales1 sales2 sales3 sales4
sales1-sales4
```

**注:** 最初のデータセット名にある数値接尾辞の先頭に 0 が置かれる場合、最後のデータセット名の数値接尾辞の桁数は最初のデータセット名の桁数と一致するか、または上回る必要があります。このように指定されていない場合はエラーが発生します。たとえば、データセットリストを sales001-sales99 または sales01-sales9 と指定するとエラーが発生します。ただし、このデータセットリストの有効な指定は、sales001-sales999 です。最初のデータセット名にある数値接尾辞の先頭に 0 が置かれない場合、最初と最後のデータセット名に使用する数値接尾辞の桁数を一致させる必要はありません。たとえば、データセットリストの指定として sales1-sales999 は有効です。

番号付きデータセットリストを使用する場合に考慮すべきその他のルールを次に示します。

- 範囲のグループを複数指定できます。

```
merge cost1-cost4 cost11-cost14 cost21-cost24;
```

- 番号付き範囲リストと名前接頭辞リストを組み合わせで指定できます。

```
merge cost1-cost4 cost2: cost33-37;
```

- 個々のデータセットとデータセットリストを組み合わせで指定できます。

```
merge cost1 cost10-cost20 cost30;
```

- データセットリストを囲んだ引用符は無視されます。

```
/* these two lines are the same */
merge sales1-sales4;
merge 'sales1'n-'sales4'n;
```

- データセット名のブランクは無効です。引用符を使用する場合、末尾にあるブランクは無視されます。

```
/* blanks in these statements will cause errors */
```

```
merge sales 1-sales 4;
merge 'sales 1'n - 'sales 4'n;
/* trailing blanks in this statement will be ignored */
merge 'sales1'n - 'sales4'n;
```

- 数値接尾辞に使用できる最大数は、2147483647 です。

```
/* this suffix will cause an error */
merge prod2000000000-prod2934850239;
```

- 物理パス名を使用できません。

```
/* physical pathnames will cause an error */
%let work_path = %sysfunc(pathname(WORK));
merge "&work_path\dept.sas7bdat"-"&work_path\emp.sas7bdat";
```

## 1対1のマージ

1対1のマージとは、複数の SAS データセットからオブザベーションを読み込んで結合し、新しいデータセットの単一のオブザベーションにする処理です。1対1のマージを実行するには、BY ステートメントを指定せずに MERGE ステートメントを使用します。MERGE ステートメントに指定したすべてのデータセットの最初のオブザベーションを結合してから、新しいデータセットに最初のオブザベーションを作成します。次に、すべてのデータセットの 2 番目のオブザベーションを結合して、新しいデータセットに 2 番目のオブザベーションを作成します。同様の作業が繰り返されます。1対1マージでは、新しいデータセットに作成されるオブザベーションの数は、MERGE ステートメントに指定した最大データセットのオブザベーションの数と一致します。1対1マージについては、例 1 を参照してください。

### 注意:

**1対1マージを使用してデータセットを結合する場合は注意が必要です。** 1対1マージでは、予期しない結果が発生する場合があります。この方法を使用する前に、データセットのサンプルを使用してプログラムのテストを実行してください。

## マッチマージ

マッチマージでは、複数の SAS データセットから、共通する変数の値に基づいてオブザベーションを取り出し、結合して、新しいデータセットの 1 つのオブザベーションにします。新しいデータセットに含まれるオブザベーションの数は、すべてのデータセットの各 BY グループに含まれるオブザベーションのうち、最も数が多いオブザベーションの合計になります。マッチマージを実行するには、MERGE ステートメントの直後に BY ステートメントを使用する必要があります。BY ステートメントに指定する変数は、すべてのデータセットに共通している必要があります。DATA ステップでは、1 つの BY ステートメントのみ、MERGE ステートメントに指定できます。MERGE ステートメントにリストされるデータセットは、BY ステートメントにリストされる変数の値に基づいて並べ替えられているか、適切なインデックスが含まれている必要があります。マッチマージについては、例 2 を参照してください。

**注:** MERGE ステートメントは、1対1のマッチマージではデカルト積を生成しません。そのかわり、MERGE ステートメントは、少なくとも 1 つのデータセット内の BY グループにオブザベーションが存在する間、1対1のマッチマージを実施します。1 つのデータセット内にある BY グループのすべてのオブザベーションを読み込んだ後、別のデータセット内にまだオブザベーションが存在する場合、特定 BY グループに関してすべてのオブザベーションが読み込まれるまで、1対1のマッチマージが実行されます。

## 比較

- MERGE ステートメントでは、複数のデータセットにあるオブザベーションが結合されます。UPDATE ステートメントでは、2つのデータセットにあるオブザベーションが結合されます。UPDATE ステートメントでは、マスターデータセットにある選択したオブザベーションの値も変更または更新されます。また、UPDATE ステートメントによりオブザベーションが追加される場合があります。
- UPDATE ステートメントと同様に、MODIFY ステートメントでも、2つのSASデータセットにあるオブザベーションが結合され、マスターデータセットにある選択したオブザベーションの値も変更または更新されます。
- 2つ以上のSET ステートメントを使用してオブザベーションを読み込んだ結果は、BY ステートメントを指定せずにMERGE ステートメントを使用してオブザベーションを読み込んだ結果と似ています。ただし、SET ステートメントを使用すると、すべてのデータセットからオブザベーションをすべて読み込む前に、オブザベーションの数が同一ではない場合は処理が中止されます。これに対して、MERGE ステートメントに指定した全データセットの全オブザベーションの処理は続けられます。

## 例

### 例 1: 1対1のマージ

次の例では、2つのデータセットにあるオブザベーションを結合し、新しいデータセットに1つのオブザベーションを作成する方法を示しています。

```
data benefits.qtr1;
  merge benefits.jan benefits.feb;
run;
```

### 例 2: マッチマージ

次の例では、2つのデータセットにあるオブザベーションをBYステートメントに指定した変数の値に基づいて結合し、新しいデータセットに1つのオブザベーションを作成する方法を示しています。

```
data inventory;
  merge stock orders;
  by partnum;
run;
```

### 例 3: データセットリストを使用したマージ

この例では、データリストを使用してマージ対象のデータセットを定義します。

```
data d008; job=3; emp=19; run;
data d009; job=3; sal=50; run;
data d010; job=4; emp=97; run;
data d011; job=4; sal=15; run;
data comb;
  merge d008-d011;
  by job;
run;
proc print data=comb;
run;
```



**例 4: BY 値と IN=データセットオプションを使用した 3 つのテーブルのマージ**

この例では、3 つのテーブル内にあるデータをマージします。

```
DATA CAFE(KEEP=NAME PLACE CNUM);
  INPUT NAME $;
  PLACE = 'CAFE ';
  CNUM = 'C' || LEFT(PUT(_N_,2.));
  DATALINES;
ANDERSON
COOPER
DIXON
FREDERIC
FREDERIC
PALMER
RANDALL
RANDALL
SMITH
SMITH
SMITH
;
RUN;

DATA VENDING (KEEP=NAME PLACE VNUM);
  INPUT NAME $;
  PLACE = 'VENDING ';
  VNUM = 'V' || LEFT(PUT(_N_,2.));
  DATALINES;
CARTER
DANIELS
GARY
GARY
HODGE
PALMER
RANDALL
RANDALL
SMITH
SMITH
SPENCER
SPENCER
SPENCER
SPENCER
;
RUN;

DATA SNACK (KEEP=NAME PLACE SNUM);
  INPUT NAME $;
  PLACE = 'SNACK ';
  SNUM = 'S' || LEFT(PUT(_N_,2.));
  DATALINES;
BARRETT
COOPER
DANIELS
DIXON
DIXON
FREDERIC
```

```

GARY
HODGE
HODGE
PALMER
RANDALL
RANDALL
SMITH
SMITH
SMITH
SMITH
SPENCER
SPENCER
;
RUN;

DATA ALL;
  MERGE CAFE(IN=CAFEIN) SNACK(IN=SNACKIN) VENDING(IN=VENDIN);
  BY NAME;
  CIN=CAFEIN; SIN=SNACKIN; VIN=VENDIN;
RUN;

PROC PRINT;
  TITLE 'MERGED DATA';
RUN;

```

MERGED DATA								
Obs	NAME	PLACE	CNUM	SNUM	VNUM	CIN	SIN	VIN
1	ANDERSON	CAFE	C1			1	0	0
2	BARRETT	SNACK		S1		0	1	0
3	CARTER	VENDING			V1	0	0	1
4	COOPER	SNACK	C2	S2		1	1	0
5	DANIELS	VENDING		S3	V2	0	1	1
6	DIXON	SNACK	C3	S4		1	1	0
7	DIXON	SNACK	C3	S5		1	1	0
8	FREDERIC	SNACK	C4	S6		1	1	0
9	FREDERIC	CAFE	C5	S6		1	1	0
10	GARY	VENDING		S7	V3	0	1	1
11	GARY	VENDING		S7	V4	0	1	1
12	HODGE	VENDING		S8	V5	0	1	1
13	HODGE	SNACK		S9	V5	0	1	1
14	PALMER	VENDING	C6	S10	V6	1	1	1
	RANDALL	VENDING	C7	S11			1	1

**例 5: BY 値と IN=データセットオプションを使用した 2 つのテーブルの  
マージ**

```
data have_a;
  input ID amount_a;
  datalines;
1 10
3 15
4 20
7 15
9 12
10 14
;

data have_b;
  input ID amount_b;
  datalines;
2 15
3 20
4 10
5 12
7 20
8 15
9 10
11 20
;

data want;
  merge have_a(in=inA) have_b(in=inB);
  by id;
  length joinType $ 2;
  joinType = cats(inA, inB);
run;

proc print data=want;
run;
quit;
```

MERGED DATA				
Obs	ID	amount_a	amount_b	joinType
1	1	10	.	10
2	2	.	15	01
3	3	15	20	11
4	4	20	10	11
5	5	.	12	01
6	7	15	20	11
7	8	.	15	01
8	9	12	10	11
9	10	14	.	10
10	11	.	20	01

### 関連項目:

#### ステートメント:

- “BY ステートメント” (33 ページ)
- “MODIFY ステートメント” (289 ページ)
- “SET ステートメント” (378 ページ)
- “UPDATE ステートメント” (408 ページ)

---

## MISSING ステートメント

入力データ内で数値データの特殊欠損値を表す文字を割り当てます。

**該当要素:** 任意の場所

**カテゴリ:** 情報

**制限事項:** このステートメントは CAS Server ではサポートされていません。

### 構文

**MISSING** *character(s)*;

### 引数

#### *character*

入力データ内で特殊欠損値を表す値です。

**範囲** 特殊欠損値には、アルファベット 26 文字(大文字または小文字)とアンダースコア(\_)を使用できます。

---

ヒント 複数の文字を指定できます。

## 詳細

通常、MISSING ステートメントは DATA ステップ内で使用されますが、ステートメントの有効範囲はグローバルです。

## 比較

MISSING=システムオプションでは、数値変数が通常の欠損値(.)の場合に出力する文字を指定できます。データに **a** や **z** などの特殊欠損値を表す文字が含まれている場合、MISSING=オプションを使用して欠損値を定義しないでください。この場合、MISSING ステートメントを使用してそれらの値を定義してください。

## 例: 特定の種類の欠損データを識別する

次の調査データのように特定の種類の欠損データを識別する必要があるとします。たとえば、データ内の **A** は回答者が調査時に不在だったことを示し、**R** は回答者が返答を拒否したことを示しています。次の例では、MISSING ステートメントを使用して、入力データ行にある値 **A** および値 **R** を、無効な数値データではなく特殊欠損値であると見なすように SAS に指示します。

```
data survey;
  missing a r;
  input id answer;
  datalines;
001 2
002 R
003 1
004 A
005 2
;
```

結果のデータセット SURVEY には、入力データで定義した値が含まれます。

## 関連項目:

### ステートメント:

- [“UPDATE ステートメント” \(408 ページ\)](#)

### システムオプション:

- [“MISSING= System Option” \(SAS Viya System Options: Reference\)](#)

---

## MODIFY ステートメント

既存のデータセットにあるオブザベーションの置き換え、削除、追加を実行します。ただし、オブザベーションのコピーは作成しません。

**該当要素:** DATA ステップ

**カテゴリ:** ファイル操作

**種類:** 実行

- 制限事項:** このステートメントは CAS Server ではサポートされていません。  
変数の追加など、SAS データセットのディスクリプタ部分を変更することはできません。
- 注:** パスワードで保護されたデータセットを変更する場合、DATA ステートメントではなく MODIFY ステートメントの適切なデータセットオプション(ALTER=または PW=) にパスワードを指定します。  
MODIFY ステートメントを使用して読み込まれた値は PDV で保持されます。読み込まれた変数のデータタイプも保持されます。詳細については、“[RETAIN ステートメント](#)” (361 ページ)を参照してください。
- 注意:** **MODIFY ステートメントを含む DATA ステップの実行中にシステムが異常終了すると、SAS データセットが破損する可能性があります。** ネイティブ SAS データファイルのオブザベーションに誤ったデータ値が含まれたり、データファイルが読み込めなくなる場合があります。ビューで参照する DBMS テーブルには影響はありません。

## 構文

- 形式 1: **MODIFY** *master-data-set* <(data-set-options)> *transaction-data-set* <(data-set-options)>  
<CUROBS=variable> <NOBS=variable> <END=variable>  
<UPDATERESET=MISSINGCHECK | NOMISSINGCHECK>;  
**BY** *by-variable*;
- 形式 2: **MODIFY** *master-data-set* <(data-set-options)> **KEY=index** </ UNIQUE>  
<KEYRESET=variable> <NOBS=variable> <END=variable>;
- 形式 3: **MODIFY** *master-data-set* <(data-set-options)>  
<NOBS=variable>POINT=variable;
- 形式 4: **MODIFY** *master-data-set* <(data-set-options)> <NOBS=variable>  
<END=variable>;

## 引数

### *master-data-set*

変更対象となる SAS データセット(マスタデータセット)を指定します。

**制限事項** このデータセットは、DATA ステートメントにも指定する必要があります。

順次アクセスまたはマッチングアクセスの場合、マスタデータセットには、SAS データファイル、SAS/ACCESS ビュー、SQL ビュー、LIBNAME ステートメントの DBMS エンジン指定できます。DATA ステップビューと SQL プロシジャのパススルー機能を使用したビューは指定できません。

POINT=オプションを指定したランダムアクセスの場合、マスタデータセットには SAS データファイルか、SAS データファイルを参照する SQL ビューを指定する必要があります。

KEY=オプションを指定したダイレクトアクセスの場合、マスタデータセットには SAS データファイルか、LIBNAME ステートメントの DBMS エンジン指定できます。SAS ファイルを指定する場合、インデックス付

きである必要があり、KEY=オプションにインデックス名を指定する必要があります。

DBMS の場合、KEY=オプションにキーワード DBKEY を設定し、インデックスとして使用する列名を DBKEY=データセットオプションに指定する必要があります。指定した列名は、DBMS に渡す WHERE 式の生成に使用されます。

ヒ データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

#### **(data-set-options)**

SAS データセット名の後ろに、1 つまたは複数の SAS データセットオプションを丸かっこで囲んで指定します。

注 データセットオプションは、処理対象のオブザベーションを DATA ステップに読み込むときに実行するアクションを指定します。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 Viya Data Set Options: Reference](#)

ヒ データセットリストに適用するデータセットオプションは、データセットリストに存在するすべてのデータセットに適用されます。

#### **transaction-data-set**

マッチングアクセスで使用する値を含む SAS データセットを指定します。この値はマスタデータセットの更新に使用されます。

制限事項 このデータセットは、DATA ステップに BY ステートメントが含まれている場合にのみ指定してください。

ヒ データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

#### **by-variable**

オブザベーションの識別に使用する、1 つまたは複数の変数名を指定します。

制限事項 VARCHAR データタイプの変数はサポートされていません。

#### **CUROBS=variable**

データセットから読み込んだオブザベーション数を含む変数を作成および名前を付与

#### **END=variable**

作成する一時変数の名前を指定します。この変数の値には終端指示子が格納されます。

制限事項 この引数は、POINT=オプションを指定した MODIFY ステートメントには使用しないでください。POINT=オプションは、MODIFY ステートメントでランダムアクセスを使用することを示しています。ランダムアクセスの場合、END=オプションに指定した変数の値が 1 に設定されることはありません。

注 この変数の初期値は 0 です。この変数の値が 1 に設定されるのは、MODIFY ステートメントによって、変更するデータセットの最後のオブザベーションが読み込まれた場合(順次アクセスの場合)、またはトランザクションデータセットの最後のオブザベーションが読み込まれた場合(マッチングアクセスの場合)です。また、この変数の値は、MODIFY ステートメントで KEY=オプションに指定した値に一致する値が見つからなかった場合にも 1 に設定されます(ランダムアクセスの場合)。

この変数はどのデータセットにも追加されません。

### KEY=index

変更する SAS データセットの単一インデックスまたは複合インデックスを指定します。KEY=オプションを指定すると、他のソースに含まれる同一の変数名のインデックス値に基づいて、SAS データセットからオブザベーションを取得します。

デ KEY=オプションの値が見つからない場合、自動変数\_ERROR\_の値は 1  
フ に設定されます。また、自動変数\_IORC\_は、SYSRC 自動呼び出しマク  
ォ ル の二ーモニク\_DSENUM に対応する値を受け入れます。“自動変  
ト 数\_IORC\_と SYSRC 自動呼び出しマクロ”(297 ページ)を参照してくだ  
さい。

制 KEY=オプションの処理は、SAS/ACCESS Engine によって異なります。  
限 KEY=オプションの処理は、SAS/ACCESS のドキュメントを参照してください。  
事  
項

VARCHAR データタイプの変数はサポートされていません。

ヒ KEYRESET=オプションを使って、読み込むデータセットのインデックス  
ン の先頭から KEY=オプションに指定された値の検索を開始するかどうか  
ト をコントロールします。

インデックス値のソースの例として、SET ステートメントに指定した別の SAS データセットや INPUT ステートメントで読み込まれた外部ファイルなどがあります。

マスターデータセットにインデックス値の重複がある場合、そのインデックス値を持つ最初のオブザベーションしか更新されません。これを回避するには、DO ループ処理を使用して、KEY=オプションに指定したデータセットの SET ステートメントがマスターデータセットの重複インデックス値に対して繰り返し実行されるようにします。

トランザクションデータセットの重複インデックス値が連続している場合、UNIQUE オプションを指定して、マスターデータセットでのインデックス値の検索が、常にインデックスの先頭から実行されるようにします。さらに、合計ステートメントを使用して、トランザクションデータセットの重複インデックス値のオブザベーションがマスターデータセットのオブザベーションに追加されるようにする必要があります。UNIQUE オプションを指定しない場合、重複インデックス値のうち、最初に検出されたトランザクションデータセットのオブザベーションのみを使用してマスターデータセットを更新します。

トランザクションデータセットの重複インデックスが連続していない場合、検索は毎回インデックスの先頭から開始されるので、重複インデックス値がマスターデータセットにそれぞれ適用されます。合計ステー



トメントを作成すると、トランザクションデータセットの重複インデックス値のオブザベーションがマスタに追加されます。

参照項目: [“KEYRESET=variable” \(293 ページ\)](#)

参照項目:

[UNIQUE \(294 ページ\)](#)

例 [“例 5: インデックスにより検索されるオブザベーションの変更” \(305 ページ\)](#)

[“例 6: 重複するインデックス値の処理” \(306 ページ\)](#)

[“例 7: I/O の制御” \(308 ページ\)](#)

### KEYRESET=variable

読み込むデータセットのインデックスの先頭から KEY=オプションに指定された値の検索を開始するかどうかをコントロールします。KEYRESET 変数の値が 1 の場合、インデックスの一番上から検索します。KEYRESET 変数の値が 0 の場合、インデックスの検索はリセットされないため、最後の検索のつぎから検索します。

制限事項 VARCHAR データタイプの変数はサポートされていません。

操作 KEYRESET=オプションは UNIQUE オプションに似ていますが、KEYRESET=オプションでは KEY = 検索をインデックスの最初からにするかどうかを決められます。

参照項目: [“KEY=index” \(292 ページ\)](#)

[“UNIQUE” \(294 ページ\)](#)

### NOBS=variable

作成する一時変数の名前を指定します。この一時変数の値は、入力データセットに含まれるオブザベーションの合計数になります。特定の SAS ビューと TAPE や XML エンジンのような順次エンジンに対しては、SAS がオブザベーション数を検知することができません。この場合、NOBS=オプションに指定した変数の値には、動作環境で使用できる最大の正の整数値が設定されます。

注 コンパイル時に、データセットのディスクリプタ情報を読み込んでから、NOBS=オプションの変数の値を自動的に割り当てます。そのため、MODIFY ステートメントの実行前にも NOBS=オプションの変数を参照できます。この変数は DATA ステップで使用できますが、新しいデータセットには追加されません。

ヒント NOBS=オプションと POINT=オプションは互いに独立しています。

例 [“例 4: オブザベーション番号により検索されるオブザベーションの変更” \(304 ページ\)](#)

### POINT=variable

オブザベーション番号をもとにランダム(ダイレクト)アクセスを使用して SAS データセットを読み込みます。variable には、読み込むオブザベシ

ンの番号を格納する変数の名前を指定します。POINT=オプションに指定した変数は DATA ステップのどの位置にでも指定できますが、SAS データセットには追加されません。

- 制限事項
- POINT=オプションは次のものとは併用できません。
- BY ステートメント
  - WHERE ステートメント
  - WHERE=データセットオプション
  - 移送形式のデータセット
  - テープまたはディスクに記録された順次データセット
  - 他社製のリレーショナルデータベース管理システムにあるテーブル

VARCHAR データタイプの変数はサポートされていません。

圧縮データセットで POINT=オプションを使用できるのは、POINTOBS=データセットオプションを YES(デフォルト値)に設定してこのデータセットを作成した場合のみです。

SAS バージョン 7 以降を使用している場合のみ、圧縮ファイルにランダムアクセス方式を使用できます。

- 要件
- POINT=オプションを使用する場合、次のプログラミング要素のどちらかまたは両方を使用してください。
- STOP ステートメント
  - POINT=オプションに指定した変数の値が無効かどうかを確認するプログラミングロジック

POINT=オプションでは指定したオブザベーションのみを読み込むため、順次アクセスでファイルを読み込む場合とは違い、ファイル終端条件を読み取ることができません。ファイル終端条件を検出すると DATA ステップを自動的に終了させるため、POINT=オプションの使用時に DATA ステップを終了させる別の方法を準備しないと、DATA ステップで無限ループが発生する場合があります。

- ヒント
- POINT=オプションに指定した変数の値がオブザベーションの番号と一致しない場合、自動変数\_ERROR\_の値が 1 に設定されます。

- 例
- “例 4: オブザベーション番号により検索されるオブザベーションの変更” (304 ページ)

## UNIQUE

変更するデータファイルの先頭から KEY=オプションに指定した値の検索を開始します。

- 制限事項
- UNIQUE オプションは、KEY=オプションを指定する場合にのみ使用できます。

- ヒント
- UNIQUE オプションは、トランザクションデータセットの中に KEY=オプションに指定したインデックス値の重複が連続して存在する場合に使用します。このオプションを使用すると、トランザクションデータセットにある重複した値ごとに、マスタデータセットでの一致する値の検索をインデックスファイルの先頭から開始します。この場合、合計ステートメントを使用する必要があります。合計ステートメントを使用しないと、重複した値による上書きが発生するため、最後のトランザクションの値のみがマスタオブザベーションに反映されます。

参照 [“KEYRESET=variable” \(293 ページ\)](#)

項目:

例 [“例 6: 重複するインデックス値の処理” \(306 ページ\)](#)

### UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK

トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある既存の変数の値を置き換えるかどうかを指定します。

#### MISSINGCHECK

トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある変数の値を置き換えないように指示します。

#### NOMISSINGCHECK

欠損値があるかどうかのチェックを実行しません。トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある変数の値を置き換えられるようにします。

デフォルト MISSINGCHECK

要件 オブザベーションの一致を検出する基準を指示する BY ステートメントに、UPDATEMODE オプションを指定する必要があります。

ヒント ただし、特殊欠損値は例外です。この場合、MISSINGCHECK が有効な場合でも、マスターデータセットの値は特殊欠損値で置き換えられます。

## 詳細

### マッチングアクセス(形式1)

マッチングアクセス方式では、BY ステートメントを使用して、トランザクションデータセットのオブザベーションとマスターデータセットのオブザベーションを突き合わせて対応させます。BY ステートメントには、トランザクションデータセットとマスターデータセットの両方に含まれる変数を指定します。

MODIFY ステートメントでトランザクションデータセットからオブザベーションを読み込むと、動的 WHERE 処理を実行し、マスターデータセットの中で対応するオブザベーションを探します。マスターデータセットのオブザベーションには、次のいずれかの処理が実行されます。

- マスターデータセットの値をトランザクションデータセットの値で置き換える
- マスターデータセットから削除する
- マスターデータセットに追加する

[“例 3: トランザクションデータセットを使用したオブザベーションの変更” \(302 ページ\)](#)には、マッチングアクセス方式の使用例が紹介されています。

### BY 値の重複(形式1)

マスターデータセットに重複が存在する場合でも、トランザクションデータセットに重複が存在する場合でも、処理に影響します。

- マスターデータセットに重複が存在する場合、重複する値のうち、最初に検出された値のみが更新されます。これは、生成される WHERE ステートメントではマスターデータセット内で常に条件に一致する最初の値を探すためです。

- トランザクションデータセットに重複が存在する場合、重複した値は上書きされます。これを回避するには、合計ステートメントを作成して重複する値をすべてマスタデータセットに追加する必要があります。合計ステートメントを使用しないと、重複している値は上書きされるので、最後のオブザベーションの値だけがマスタデータセットのオブザベーションに反映されます。

### インデックス付きの値によるダイレクトアクセス(形式2)

この方法では、MODIFY ステートメントに KEY=オプションを指定する必要があります。KEY=オプションには、変更するデータセットに含まれるインデックス付きの変数の名前を指定する必要があります。もう 1 つのデータソース(通常、SET ステートメントに指定した SAS データセットか、INPUT ステートメントで読み込んだ外部ファイル)から、インデックス付きの変数に値を提供する同じ名前の変数を得ます。MODIFY ステートメントは、インデックス値を使用して変更対象のデータセットに格納されているオブザベーションを検出します。

“例 5: インデックスにより検索されるオブザベーションの変更” (305 ページ)では、インデックス付きの値を使用したダイレクトアクセス方式の例を紹介しています。

### インデックス値の重複(形式2)

- マスタデータセットにインデックス付きの変数の値が重複して存在する場合、最初に検出された値のみを対象にして取得、変更、置き換えを実行します。DO ループ処理を使用して KEY=オプションを指定した SET ステートメントを繰り返し実行すると、重複するすべての値がトランザクションの値で更新されます。
- データソースの同一名の変数に重複した値が連続せずに存在する場合、MODIFY ステートメントは、データソースのインデックス値に一致する値が最初に検出されたマスタデータセットのオブザベーションに対して、重複するトランザクションを繰り返し適用します。そのため、重複するトランザクションの最後の値のみがマスタオブザベーションに反映されます。これを回避するには、合計ステートメントを作成し、重複するトランザクションのそれぞれの値をマスタオブザベーションに追加する必要があります。
- データソースの同一名の変数に重複した値が連続して存在する場合、データソースの中で最初に検出されたオブザベーションの値がマスタデータセットに適用されます。ただし、重複する値のうち、データソースの中で 2 番目に検出された値に対応するオブザベーションをマスタデータセットで探そうとすると、エラーが発生するため DATA ステップが終了します。このエラーを回避するには、MODIFY ステートメントに UNIQUE オプションを指定します。UNIQUE オプションを使用すると、マスタデータセットの先頭に戻ってから一致するインデックス値を取得します。この場合、合計ステートメントを記述して、重複する値をすべて合計する必要があります。合計ステートメント記述しない場合、最後に検出された値のみがマスタオブザベーションに反映されます。

“例 6: 重複するインデックス値の処理” (306 ページ)では、重複するインデックス値を扱う例を紹介しています。

- 両方のデータセットに重複するインデックス値が存在する場合、SQL を使用すると、トランザクションデータセットの重複した値をマスタデータセットの重複した値と 1 対 1 で対応させることができます。

### オブザベーション番号によるダイレクトアクセス(ランダムアクセス)(形式3)

MODIFY ステートメントに POINT=オプションを使用すると、もう 1 つのデータソース(マスタデータセットを除く)の変数名を指定することができます。この変

数の値がマスターデータセットで変更するオブザベーションの番号になります。MODIFY ステートメントは POINT=オプションに指定した変数の値を使用して、データセットに格納されている変更対象のオブザベーションを取得します。(圧縮データセットに POINT=オプションを使用できるのは、データセットが POINTOBS=データセットオプションを指定して作成された場合に限られます。)

プログラミング時には、POINT=オプションの変数の値を検証すること、また、自動変数\_ERROR\_の値のステータスを確認することをお勧めします。

“例 4: オブザベーション番号により検索されるオブザベーションの変更” (304 ページ)では、オブザベーション番号によるダイレクトアクセス(ランダムアクセス)方式について説明しています。

**注意:**

**POINT=オプションを指定すると、無限ループが発生する可能性があります。** POINT=オプションを使用する場合は、DATA ステップを正常に停止できないと、DATA ステップで無限ループが発生する可能性があるため注意してください。STOP ステートメントを使用するか、POINT=オプションに指定した変数の値が無効かどうかを確認するプログラミングロジックを使用するか、またはこの両方を使用してください。

#### 順次アクセス(形式 4)

順次アクセス方式は、MODIFY ステートメントで最も簡単な形式ですが、ダイレクトアクセス方式に比べると制御できる要素は少なくなります。順次アクセス方式を使用する場合、NOBS=オプションおよび END=オプションを使用してデータセットを変更できます。ただし、POINT=オプションまたは KEY=オプションは使用しません。

#### 自動変数\_IORC\_と SYSRC 自動呼び出しマクロ

自動変数\_IORC\_には、MODIFY ステートメントで I/O 操作を実行するたびにリターンコードが格納されます。自動変数\_IORC\_の値を検証するもっとも簡単な方法は、SYSRC 自動呼び出しマクロで提供されるニーモニックコードを使用することです。それぞれのニーモニックコードには、特定の条件が記述されています。そのため、ニーモニックを使用すると、DATA ステッププログラムで発生した問題を簡単に検証できるようになります。次のコードを使用すると便利です。

##### \_DSENMR

トランザクションデータセットのオブザベーションがマスターデータセット上に存在しないことを示します(MODIFY ステートメントと BY ステートメントを併用した場合にのみ使用)。異なる BY 値を持つ連続するオブザベーションがマスターデータセットに存在しない場合、どちらのオブザベーションにも \_DSENMR が返されます。

##### \_DSEMTR

指定した BY 値を持つ複数のトランザクションデータセットのオブザベーションがマスターデータセット上に存在しないことを示します(MODIFY ステートメントと BY ステートメントを併用した場合にのみ使用)。同じ BY 値を持つ連続したオブザベーションがマスターデータセットに存在しない場合、最初のオブザベーションに対しては \_DSENMR、後続のオブザベーションに対しては \_DSEMTR が返されます。

##### \_DSENOM

変更対象のデータセットに KEY=オプションまたは POINT=オプションで要求したオブザベーションが含まれていないことを示します。



**\_SENOCHN**

オブザベーションに対して OUTPUT ステートメントまたは REPLACE ステートメントを実行しようとしています。このオブザベーションに含まれているキー値と同じ値が、一意のキー値が必要な既存のインデックス付きデータセットに存在することを示しています。

**\_SOK**

オブザベーションが検出されたことを示しています。

注: IORCMMSG 関数は、自動変数 `_IORC_` の現在の値に関連するエラーメッセージを指定された形式で返します。

“例 7: I/O の制御” (308 ページ) では、自動変数 `_IORC_` および `SYSRC` 自動呼び出しマクロの使用方法について説明しています。

## DATA ステップで MODIFY が使用される場合のオブザベーションの書き込み

DATA ステップに MODIFY ステートメントが含まれている場合にオブザベーションを SAS データセットに書き出す方法は、他にどのステートメントが使用されているかによって異なります。次の方法が考えられます。

### 明示的なステートメントなし

現在のオブザベーションを SAS データセットの元の位置に書き込みます。このアクションは、DATA ステップの最後のアクションとして実行されます (REPLACE ステートメントが DATA ステップの最後のステートメントとして記述されているのと同じように処理)。

### OUTPUT ステートメント

OUTPUT ステートメントにデータセットが指定されていない場合、DATA ステップに指定したすべてのデータセットの最後に現在のオブザベーションを書き込みます。データセットが指定されている場合、このステートメントは指定されたデータセットの最後に現在のオブザベーションを書き込みます。このアクションは、DATA ステップで OUTPUT ステートメントが検出された時点で実行されます。

### REPLACE <data-set-name>ステートメント

指定された 1 つまたは複数のデータセットに現在のオブザベーションを再度書き込みます。また、引数を指定しない場合は、DATA ステートメントに指定されている各データセットに現在のオブザベーションを再度書き込みます。このアクションは、REPLACE ステートメントが検出された時点で実行されます。

### REMOVE <data-set-name>ステートメント

指定された 1 つまたは複数のデータセットから現在のオブザベーションを削除します。また、引数を指定しない場合は、DATA ステートメントに指定されている各データセットから現在のオブザベーションを削除します。データセットを管理しているエンジンの特性に応じて、削除が物理的に実行される場合も、論理的に実行される場合もあります。

これらのステートメントを使用する場合は、次に注意してください。

- OUTPUT ステートメント、REPLACE ステートメント、REMOVE ステートメントがどれも指定されていない場合、デフォルトのアクションは REPLACE ステートメントになります。
- OUTPUT、REPLACE、REMOVE の各ステートメントは互いに独立して動作します。1 つのオブザベーションに適用する OUTPUT、REPLACE、REMOVE の各ステートメントを複数記述することができます。ただし、いったん OUTPUT、REPLACE、REMOVE のいずれかのステートメントを実行した場合

は、次の REPLACE ステートメントまたは REMOVE ステートメントを実行する前に、MODIFY ステートメントを再度実行する必要があります。

次の例の条件ロジックに示すように、OUTPUT ステートメントおよび REPLACE ステートメントを使用できます。これは、オブザベーションごとに REPLACE ステートメントまたは OUTPUT ステートメントのどちらか 1 つのみが実行されるためです。

```
data master;
  modify master trans; by key;
  if _iorc_=0 then replace;
  else
    output;
run;
```

ただし、この例に示すように、同じオブザベーションに対して複数の REPLACE 操作を実行しないでください。

```
data master;
  modify master;
  x=1;
  replace;
  replace;
run;
```

オブザベーションごとに複数の OUTPUT ステートメントを記述できます。ただし、複数の OUTPUT ステートメントを使用する場合は注意が必要です。OUTPUT ステートメントを 1 つだけ使用する場合でも、無限ループが発生する可能性があります。

```
data master;
  modify master;
  output;
run;
```

- DATA ステップで OUTPUT、REPLACE、REMOVE のいずれかのステートメントを使用すると、オブザベーションのデフォルトの置き換えより優先されません。DATA ステップでこれらのステートメントのいずれかを使用する場合は、実行するアクションを明示的に記述する必要があります。
- OUTPUT ステートメントと REPLACE ステートメントの両方、または REMOVE ステートメントを指定したオブザベーションに対して実行する場合、オブザベーションポインタの位置を正しく保つために OUTPUT 操作を最後に実行する必要があります。

“例 8: オブザベーションの置換と削除、オブザベーションの他の SAS データセットへの書き込み” (310 ページ) では、OUTPUT、REMOVE、REPLACE の各ステートメントを使用してオブザベーションを書き込む方法について説明しています。

### 欠損値と MODIFY ステートメント

デフォルトでは、UPDATEMODE=MISSINGCHECK オプションが有効です。そのため、トランザクションデータセットの欠損値によって、マスタデータセットの既存の値が置き換えられることはありません。そのため、すべての変数ではなく、一部の変数のみを更新する場合、またオブザベーションごとに更新する変数が異なる場合は、変更しない変数を欠損値に設定します。トランザクションデータに含まれる欠損値でマスタデータセットに含まれる既存の値を置き換える場合は、UPDATEMODE=NOMISSINGCHECK を使用します。

UPDATEMODE=MISSINGCHECK が有効な場合でも、トランザクションデータセットで特殊欠損値の文字を使用すると、既存の値を欠損値に置き換えることがで

きます。特殊欠損値を含むトランザクションデータセットを作成するには、DATA ステップで MISSING ステートメントを使用します。トランザクションデータセットに **A** から **Z** までの特殊欠損値の 1 つを定義すると、マスタデータセットに含まれる数値変数はその値に更新されます。

マスタデータセットの値を通常の欠損値に設定する場合は、トランザクションデータセット内でアンダースコア(\_)を 1 つ使用して欠損値を示します。このように指定すると、マスタデータセットの値は、数値の欠損値がピリオド(.)に設定され、文字の欠損値は空白に設定されます。

特殊欠損値文字の定義方法および使用方法の詳細については、“MISSING ステートメント”(288 ページ)を参照してください。

### データセットオプションを用いた MODIFY の使用

プログラムでデータセットオプション(KEEP=オプションなど)を使用する場合は、MODIFY ステートメントでマスタデータセットに対してこれらのオプションを指定します。DATA ステートメントでデータセットオプションを使用すると、予期しない結果が生じることがあります。

## 比較

- DATA ステップで MERGE、SET、UPDATE のいずれかのステートメントを使用すると、新しい SAS データセットが作成されます。新しいデータセットのデータセットディスクリプタは、元のデータセットのディスクリプタとは異なります(変数の追加、削除、ラベルの変更などによる)。ただし、DATA ステップで MODIFY ステートメントを使用する場合は、新しいデータセットは作成されません。そのため、データセットディスクリプタが変更されることはありません。

DBMS の置き換えルールの詳細については、SAS/ACCESS のマニュアルを参照してください。

- MODIFY ステートメントを BY ステートメントと併用する場合、MODIFY ステートメントは UPDATE ステートメントとほとんど同じように機能します。ただし、次の場合は除きます。
  - マスタデータセットとトランザクションデータセットのどちらも並び替えやインデックスの作成を行う必要はありません。(BY ステートメントを指定した MODIFY ステートメントでは、動的 WHERE 処理がトリガされます。)

注: 並べ替えまたはインデックスの作成を実行していない SAS データセットを MODIFY ステートメントで変更する場合、動的 WHERE 処理はシステムに負担をかけることがあります。マスタデータセットで並べ替えまたはインデックスの作成を実行しておく、特にサイズが大きいファイルの場合には処理時間を削減できます。

- 重複した BY 変数の値を持つオブザベーションがマスタデータセットとトランザクションデータセットの両方に存在する場合があります。MODIFY ステートメントでの重複する値の処理方法については、“BY 値の重複(形式 1)”(295 ページ)を参照してください。
- MODIFY ステートメントでは、UPDATE ステートメントと同じように、データセットのディスクリプタ情報を変更することはできません。そのため、このステートメントでは、変数の追加や削除、変数ラベルの変更などは実行できません。



## 例

### 例 1: 例で使用する入力データセットの説明

例では、INVTY.STOCK データセットを変更します。INVTY.STOCK データセットには、次の変数が含まれています。

#### PARTNO

文字変数です。個々のツール番号を識別する一意の値が格納されます。

#### DESC

文字変数です。各ツールのテキストでの説明が格納されます。

#### INSTOCK

数値変数です。社内で保有している各ツールの在庫数が格納されます。

#### RECDATE

数値変数です。どの変数 INSTOCK の値が現在のものかを示す SAS 日付の値が格納されます。

#### PRICE

数値変数です。各ツールの単価が格納されます。

さらに、データセット INVTY.STOCK には、変数 PARTNO の値をキーとする単一インデックスが格納されています。次の DATA ステップを実行すると、データセット INVTY.STOCK が作成されます。

```
libname invty 'SAS-library';

data invty.stock(index=(partno));
  input PARTNO $ DESC $ INSTOCK @17
        RECDATE date7. @25 PRICE;
  format recdate date7.;
  datalines;
K89R seal 34 27jul95 245.00
M4J7 sander 98 20jun95 45.88
LK43 filter 121 19may96 10.99
MN21 brace 43 10aug96 27.87
BC85 clamp 80 16aug96 9.55
NCF3 valve 198 20mar96 24.50
KJ66 cutter 6 18jun96 19.77
UYN7 rod 211 09sep96 11.55
JD03 switch 383 09jan97 13.99
BV1E timer 26 03jan97 34.50
;
```

### 例 2: すべてのオブザベーションの変更

この例では、データセット INVTY.STOCK のすべてのレコードにある日付を現在の日付で置き換えます。また、データセット INVTY.STOCK に格納されているすべてのオブザベーションに対して、変数 RECDATE の値を現在の日付に置き換えます。

```
data invty.stock;
  modify invty.stock;
  recdate=today();
run;
proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```

## アウトプット 2.19 RECDATE フィールドの更新結果

INVTY.STOCK				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	34	08DEC10	245.00
M4J7	sander	98	08DEC10	45.88
LK43	filter	121	08DEC10	10.99
MN21	brace	43	08DEC10	27.87
BC85	clamp	80	08DEC10	9.55
NCF3	valve	198	08DEC10	24.50
KJ66	cutter	6	08DEC10	19.77
UYN7	rod	211	08DEC10	11.55
JD03	switch	383	08DEC10	13.99
BV1E	timer	26	08DEC10	34.50

MODIFY ステートメントは、更新処理を行うためにデータセット INVTY.STOCK を開きます。DATA ステップの1回の繰り返しで、データセット INVTY.STOCK のオブザベーションを1つ読み込みます。また、コードに記述された処理を実行します。この場合、DATA ステップを繰り返すたびに、変数 RECDATE の値を TODAY 関数の結果で置き換えます。DATA ステップの最後に暗示的な REPLACE ステートメントを指定すると、各オブザベーションがデータセット INVTY.STOCK の元の位置に書き込まれます。

**例 3: トランザクションデータセットを使用したオブザベーションの変更**

この例では、新しく受け取った在庫数をデータセット INVTY.STOCK に追加し、在庫を受け取った日付を更新します。WORK ライブラリにあるトランザクションデータセット ADDINV には新しいデータが含まれています。

ADDINV データセットは、更新された情報を格納するデータセットです。また、ADDINV データセットには、次の変数が格納されています。

**PARTNO**

文字変数です。データセット INVTY.STOCK のインデックス付き変数 PARTNO に対応しています。

**NWSTOCK**

数値変数です。新しく受け取った各ツールの在庫数を示しています。

データセット ADDINV は、MODIFY ステートメントに指定されている2番目のデータセットです。そのため、データセット ADDINV をトランザクションデータセットとして使用し、このデータセットの各オブザベーションを順番に読み込みます。BY ステートメントには共通する変数 PARTNO が指定されています。そのため、MODIFY ステートメントは、データセット ADDINV の変数 PARTNO の値に最初に一致する値を、データセット INVTY.STOCK の変数 PARTNO の値から探します。一致する値が検出されたオブザベーションごとに、DATA ステップによって変数 RECDATE の値が本日の日付に変更されます。また、変数 INSTOCK の値がこれまでの変数 INSTOCK の値と ADDINV に格納されている変数 NWSTOCK の値の合計に置き換えられます。MODIFY ステートメントでは、

データセットディスクリプタが変更されるので、変数 NWSTOCK はデータセット INVTY.STOCK に追加されません。そのため、DROP ステートメントで変数 NWSTOCK を指定する必要はありません。

この例では、データセット ADDINV をトランザクションデータセットとして指定します。このデータセットにはデータセット INVTY.STOCK を変更するための情報が格納されています。共通する変数を BY ステートメントに指定します。この変数の値を基にして、データセット INVTY.STOCK に格納されているオブザベーションを検索します。

次の DATA ステップを実行すると、データセット ADDINV が作成されます。

```
data addinv;
  input PARTNO $ NWSTOCK;
  datalines;
K89R 55
M4J7 21
LK43 43
MN21 73
BC85 57
NCF3 90
KJ66 2
UYN7 108
JD03 55
BV1E 27
;
```

次の DATA ステップでは、データセット ADDINV の値を使用してデータセット INVTY.STOCK を更新します。

```
libname invty 'SAS-library';

data invty.stock;
  modify invty.stock addinv;
  by partno;
  RECDATE=today();
  INSTOCK=instock+nwstock;
  if _iorc_=0 then replace;
run;

proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```

## アウトプット 2.20 INSTOCK フィールドおよび RECDATE フィールドの更新結果

INVTY.STOCK				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	89	08DEC10	245.00
M4J7	sander	119	08DEC10	45.88
LK43	filter	164	08DEC10	10.99
MN21	brace	116	08DEC10	27.87
BC85	clamp	137	08DEC10	9.55
NCF3	valve	288	08DEC10	24.50
KJ66	cutter	8	08DEC10	19.77
UYN7	rod	319	08DEC10	11.55
JD03	switch	438	08DEC10	13.99
BV1E	timer	53	08DEC10	34.50

**例 4: オブザベーション番号により検索されるオブザベーションの変更**

この例では、データセット NEWP を読み込みます。次に、データセット INVTY.STOCK の中で更新対象とするオブザベーション番号を TOOL\_OBS の値に基づいて特定してから、更新を実行します。ここでは、変数 PRICE の値を変数 NEWP の値で置き換えるため、割り当てステートメントを使用して更新作業を明示的に指定しています。

データセット NEWP には、次の 2 つの変数が格納されています。

**TOOL\_OBS**

ツール会社のマスタデータセット INVTY.STOCK にある各ツールのオブザベーション番号が格納されています。

**NEWP**

各ツールの新価格が格納されています。

次の DATA ステップを実行すると、データセット NEWP が作成されます。

```
data newp;
  input TOOL_OBS NEWP;
  datalines;
1 251.00
2 49.33
3 12.32
4 30.00
5 15.00
6 25.75
7 22.00
8 14.00
9 14.32
10 35.00
;
```

次の DATA ステップを実行すると、データセット INVTY.STOCK が更新されます。

```
libname invty 'SAS-library';

data invty.stock;
  set newp;
  modify invty.stock point=tool_obs
    nobs=max_obs;
  if _error_=1 then
    do;
      put 'ERROR occurred for TOOL_OBS=' tool_obs /
        'during DATA step iteration' _n_ /
        'TOOL_OBS value might be out of range.';
      _error_=0;
      stop;
    end;
  PRICE=newp;
  RECDATE=today();
run;

proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```

**アウトプット 2.21** RECDATE フィールドおよび PRICE フィールドの更新結果

INVTY.STOCK				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	34	08DEC10	251.00
M4J7	sander	98	08DEC10	49.33
LK43	filter	121	08DEC10	12.32
MN21	brace	43	08DEC10	30.00
BC85	clamp	80	08DEC10	15.00
NCF3	valve	198	08DEC10	25.75
KJ66	cutter	6	08DEC10	22.00
UYN7	rod	211	08DEC10	14.00
JD03	switch	383	08DEC10	14.32
BV1E	timer	26	08DEC10	35.00

### 例 5: インデックスにより検索されるオブザベーションの変更

この例では、KEY=オプションを使用して、データセット ADDINV の変数 PARTNO の値とデータセット INVTY.STOCK の変数 PARTNO のインデックス付きの値で一致する値を探し、取得するオブザベーションを特定します。データセット ADDINV の作成については、“[例 3: トランザクションデータセットを使用したオブザベーションの変更](#)” (302 ページ)を参照してください。

KEY=オプションによって、MODIFY ステートメントで更新対象のオブザベーションに直接アクセスできるインデックス値が指定されます。動的 WHERE 処理は実行されません。この例では、マスタデータセット INVTY.STOCK の変数 INSTOCK の値に、トランザクションデータセット ADDINV の変数 NWSTOCK の値を追加します。

```
libname invty 'SAS-library';

data invty.stock;
  set addinv;
  modify invty.stock key=partno;
  INSTOCK=instock+nwstock;
  RECDATE=today();
  if _iorc_=0 then replace;
run;

proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```

**アウトプット 2.22** インデックスを使用した INSTOCK フィールドと RECDATE フィールドの更新結果

INVTY.STOCK				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	89	08DEC10	245.00
M4J7	sander	119	08DEC10	45.88
LK43	filter	164	08DEC10	10.99
MN21	brace	116	08DEC10	27.87
BC85	clamp	137	08DEC10	9.55
NCF3	valve	288	08DEC10	24.50
KJ66	cutter	8	08DEC10	19.77
UYN7	rod	319	08DEC10	11.55
JD03	switch	438	08DEC10	13.99
BV1E	timer	53	08DEC10	34.50

### 例 6: 重複するインデックス値の処理

この例では、SET データセットに格納されている重複した変数の値を MODIFY ステートメントで処理する方法を示します。この SET データセットによって、マスタデータセットのインデックス値が提供されます。

データセット NEWINV は、更新された情報が格納するデータセットです。また、データセット NEWINV には、次の変数が格納されています。

#### PARTNO

文字変数です。データセット INVTY.STOCK のインデックス付き変数 PARTNO に対応しています。データセット NEWINV では、変数 PARTNO に重複した値が存在しています。この場合、**M4J7** が 2 つ含まれています。

## NWSTOCK

数値変数です。新しく受け取った各ツールの在庫数を示しています。

次の DATA ステップを実行すると、データセット NEWINV が作成されます。

```
data newinv;
  input PARTNO $ NWSTOCK;
  datalines;
K89R 55
M4J7 21
M4J7 26
LK43 43
MN21 73
BC85 57
NCF3 90
KJ66 2
UYN7 108
JD03 55
BV1E 27
;
```

次を実行すると、データセット NEWINV で 2 番目に検出される **M4J7** に一致するオブザベーションをデータセット INVTY.STOCK で探そうとしたときにエラーが発生し、DATA ステップが異常終了します。

```
libname invty 'SAS-library';

/* This DATA step terminates with an error! */
data invty.stock;
  set newinv;
  modify invty.stock key=partno;
  INSTOCK=instock+nwstock;
  RECDATE=today();
run;
```

SAS ログには次のメッセージが表示されます。

```
ERROR: No matching observation was found in MASTER data set.PARTNO=M4J7 NWSTOCK=26
DESC=sander INSTOCK=166 RECDATE=08DEC10 PRICE=45.88 _ERROR_=1 _IORC_=1230015 _N_=3 NOTE:
The SAS System stopped processing this step because of errors.NOTE: There were 3 observations read
from the data set WORK.NEWINV.NOTE: The data set INVTY.STOCK has been updated.There were 2
observations rewritten, 0 observations added and 0 observations deleted.
```

MODIFY ステートメントに UNIQUE オプションを追加すると、前述の DATA ステップで発生したエラーを回避することができます。UNIQUE オプションを追加すると、SET データセットで一致する値を検索するたびに、インデックスの先頭に戻ってから検索を実行します。そのため、SET データセットで検出されるそれぞれの **M4J7** に対して、マスタデータセットに含まれる **M4J7** が検出されます。出力される **M4J7** >の更新結果では、データセット NEWINV にある変数 NWSTOCK の 2 つの **M4J7** の値が、データセット INVTY.STOCK の変数 INSTOCK の値 **M4J7** に追加されます。次に、合計ステートメントで値を合計します。合計ステートメントを指定しない場合、**M4J7** の最後のインスタンスの値だけがデータセット INVTY.STOCK に反映されます。

```
data invty.stock;
  set newinv;
  modify invty.stock key=partno / unique;
  INSTOCK=instock+nwstock;
  RECDATE=today();
  if _iorc_=0 then replace;
```

```
run;
proc print data=invty.stock noobs;
  title 'Results of Using the UNIQUE Option';
run;
```

**アウトプット 2.23** UNIQUE オプションを使用した INSTOCK フィールドと RECDATE フィールドの更新結果

PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	89	08DEC10	245.00
M4J7	sander	145	08DEC10	45.88
LK43	filter	164	08DEC10	10.99
MN21	brace	116	08DEC10	27.87
BC85	clamp	137	08DEC10	9.55
NCF3	valve	288	08DEC10	24.50
KJ66	cutter	8	08DEC10	19.77
UYN7	rod	319	08DEC10	11.55
JD03	switch	438	08DEC10	13.99
BV1E	timer	53	08DEC10	34.50

### 例 7: I/O の制御

この例では、SYSRC 自動呼び出しマクロと自動変数\_IORC\_を使用して、I/O 条件を制御しています。これにより、検出されない可能性のある予期しない結果を防ぐことができます。この例では、インデックス値を使用したダイレクトアクセス方式で、データセット INVTY.STOCK を更新します。データセット NEWSHIP のデータにより、データセット INVTY.STOCK が更新されます。

次の DATA ステップを実行すると、データセット NEWSHIP が作成されます。

```
data newship;
  input PARTNO $ DESC $ NWSTOCK @17
        SHPDATE date7. @25 NWPRICE;
  datalines;
K89R seal 14 14nov96 245.00
M4J7 sander 24 23aug96 47.98
LK43 filter 11 29jan97 14.99
MN21 brace 9 09jan97 27.87
BC85 clamp 12 09dec96 10.00
ME34 cutter 8 14nov96 14.50
;
```

SELECT ステートメントに指定したそれぞれの WHEN 句では、SYSRC 自動呼び出しマクロが返す入出力のリターンコードごとに対応するアクションを指定しています。

- \_SOK は、MODIFY ステートメントが正常に実行されたことを示します。



- `_DSENUM` は、データセット `INVTY.STOCK` の中に一致するオブザベーションが見つからなかったことを示します。OUTPUT ステートメントでは、データセット `INVTY.STOCK` にオブザベーションを追加するように指定しています。出力結果に表示される最後のオブザベーションを確認してください。
- `SYSRC` 自動呼び出しマクロで他のコードが返された場合は、DATA ステップを終了し、PUT ステートメントでメッセージをログに出力します。

```
libname invty 'SAS-library';

data invty.stock;
  set newship;
  modify invty.stock key=partno;
  select (_iorc_);
  when (%sysrc(_sok)) do;
    INSTOCK=instock+nwstock;
    RECDATE=shpdate;
    PRICE=nwprice;
    replace;
  end;
  when (%sysrc(_dsenom)) do;
    INSTOCK=nwstock;
    RECDATE=shpdate;
    PRICE=nwprice;
    output;
    _error_=0;
  end;
  otherwise do;
    put
      'An unexpected I/O error has occurred.!'
      'Check your data and your program';
    _error_=0;
    stop;
  end;
end;

run;

proc print data=invty.stock noobs;
  title 'INVTY.STOCK Data Set';
run;
```

## アウトプット 2.24 更新後の INVTY.STOCK データセット

PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	48	14NOV96	245.00
M4J7	sander	122	23AUG96	47.98
LK43	filter	132	29JAN97	14.99
MN21	brace	52	09JAN97	27.87
BC85	clamp	92	09DEC96	10.00
NCF3	valve	198	20MAR96	24.50
KJ66	cutter	6	18JUN96	19.77
UYN7	rod	211	09SEP96	11.55
JD03	switch	383	09JAN97	13.99
BV1E	timer	26	03JAN97	34.50
ME34	cutter	8	14NOV96	14.50

**例 8: オブザベーションの置換と削除、オブザベーションの他の SAS データセットへの書き込み**

この例では、オブザベーションの置き換えや削除、別のデータセットへのオブザベーションの書き込みが実行できることを示しています。この例に示すように、OUTPUT ステートメント、REPLACE ステートメント、REMOVE ステートメントを使用する場合、デフォルトのステートメントが生成されないため、実行するアクションを明示的に指定する必要があります。

1997 年に受け取った部品はデータセット INVTY.STOCK97 に書き込み、データセット INVTY.STOCK から削除します。同じように 1995 年に受け取った部品はデータセット INVTY.STOCK95 に書き込み、データセット INVTY.STOCK から削除します。1996 年に受け取った部品だけはデータセット INVTY.STOCK に残り、データセット INVTY.STOCK の中でだけ PRICE の値を更新します。

```
libname invty 'SAS-library';

data invty.stock invty.stock95 invty.stock97;
  modify invty.stock;
  if reccdate>'01jan97'd then do;
    output invty.stock97;
    remove invty.stock;
  end;
  else if reccdate<'01jan96'd then do;
    output invty.stock95;
    remove invty.stock;
  end;
  else do;
    price=price*1.1;
    replace invty.stock;
```

```

end;
run;

proc print data=invty.stock noobs;
  title 'New Prices for Stock Received in 1996';
run;

```

アウトプット 2.25 特定の SAS データセットにオブザベーションを書き込んだ後の出力結果

New Prices for Stock Received in 1996				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
LK43	filter	121	19MAY96	12.089
MN21	brace	43	10AUG96	30.657
BC85	clamp	80	16AUG96	10.505
NCF3	valve	198	20MAR96	26.950
KJ66	cutter	6	18JUN96	21.747
UYN7	rod	211	09SEP96	12.705

**関連項目:**

- [SAS Viya SQL Procedure User's Guide](#)

**ステートメント:**

- [“MISSING ステートメント” \(288 ページ\)](#)
- [“OUTPUT ステートメント” \(314 ページ\)](#)
- [“REMOVE ステートメント” \(354 ページ\)](#)
- [“REPLACE ステートメント” \(357 ページ\)](#)
- [“UPDATE ステートメント” \(408 ページ\)](#)

---

**マルチステートメント**

データの終わりを示すか、またはプレースホルダとして動作します。

**該当要素:** 任意の場所

**カテゴリ:** アクション

CAS

**種類:** 実行

**構文**

```

;
or

```

◆◆

### 引数なし

ヌルステートメントは、プログラム内でデータ行の終わりを示します。

### 詳細

基本的な使用方法として、ヌルステートメントは DATALINES ステートメントまたは CARDS ステートメントの後に続くデータ行の終わりを示します。この場合、ヌルステートメントはステップ境界として動作します。データ行にセミコロンが含まれている場合は、DATALINES4 ステートメントまたは CARDS4 ステートメントを使用し、セミコロンを 4 つ使用するヌルステートメントでデータ行の終わりを示します。

ヌルステートメントはアクションを実行しませんが、実行ステートメントの 1 つです。そのため、ヌルステートメントの前にラベルを指定したり、このステートメントを条件付き処理で使用できます。

### 例: データ行の終わりを示す

- このプログラムのヌルステートメントはデータ行の終わりを示しています。また、ステップ境界として動作します。

```
data test;
  input score1 score2 score3;
  datalines;
55 135 177
44 132 169
;
```

- 次の入力データレコードにはセミコロンが含まれています。DATALINES4 ステートメントに続けてヌルステートメントを使用し、データ行の終わりを示します。

```
data test2;
  input code1 $ code2 $ code3 $;
  datalines4;
55;39;1 135;32;4 177;27;3
78;29;1 149;22;4 179;37;3
◆◆
```

- ヌルステートメントはプログラムの開発中に使用すると便利です。たとえば、ステートメントラベルの後にヌルステートメントを指定すると、ラベルの後のステートメントを記述する前にプログラムをテストすることができます。

```
data _null_;
  set dsn;
  file print header=header;
  put 'report text';
  ...more statements...
  return;
  header;;
run;
```

## 関連項目:

### ステートメント:

- [“DATALINES ステートメント” \(57 ページ\)](#)
- [“DATALINES4 ステートメント” \(59 ページ\)](#)
- [“GO TO ステートメント” \(132 ページ\)](#)
- [“LABEL ステートメント” \(217 ページ\)](#)

---

## OPTIONS ステートメント

1 つまたは複数の SAS システムオプションの値の指定や変更を行います。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

## 構文

**OPTIONS** *option(s)*;

## 引数

### *option*

変更する SAS システムオプションを 1 つ以上指定します。

## 詳細

OPTIONS ステートメントで実行される変更は、ジョブ、セッション、SAS プロセスを終了するまで、または他の OPTIONS ステートメントでオプションを再度変更するまで有効となります。SAS システムオプションは、OPTIONS ステートメントを通じて指定できます。また、SAS の起動時や SAS プロセスの初期化時にも指定できます。

サイト管理者によって制限されているオプションを設定しようとすると、オプションは規制されていて変更はできないというメッセージを出力します。詳細については、[“Restricted Options” \(SAS Viya System Options: Reference\)](#)を参照してください。

**注:** すべての SAS ジョブまたはセッションで特定のオプショングループを有効にするには、autoexec ファイルに OPTIONS ステートメントを保存するか、構成ファイルまたは custom\_option\_set にシステムオプションを記述します。

**注:** システムオプションがヌル値の場合、GETOPTION 関数を実行すると、' '(一重引用符で囲まれた 1 つのブランク)の値を返します(例:EMAILID=' ')。この GETOPTION の値は、OPTIONS ステートメントに使用されます。

OPTIONS ステートメントは、データ行の中を除き、SAS プログラム内のどの位置にも配置できます。

## 例: システムオプションの値の変更

次の例では、通常 SAS 出力にある日付を書き出さないように指定し、行サイズを 72 に設定します。

```
options nodate linesize=72;
```

### 関連項目:

[“Definition of System Options” \(SAS Viya System Options: Reference\)](#)

---

## OUTPUT ステートメント

現在のオブザベーションを SAS データセットに書き込みます。

**該当要素:** DATA ステップ  
**カテゴリ:** アクション  
CAS  
**種類:** 実行

---

### 構文

```
OUTPUT <data-set-name(s)>;
```

#### 引数なし

引数を指定せずに OUTPUT を使用すると、現在のオブザベーションが DATA ステートメントに指定したすべてのデータセットに書き込まれます。

MODIFY ステートメントが存在する場合、引数を指定せずに OUTPUT ステートメントを使用すると、現在のオブザベーションが MODIFY ステートメントに指定したデータセットの最後に書き込まれます。

#### 引数

##### *data-set-name*

オブザベーションを書き込むデータセットの名前を指定します。

**制限事項** OUTPUT ステートメントに指定したすべての名前は、DATA ステートメントにも指定する必要があります。

**ヒント** データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

OUTPUT ステートメントには、DATA ステップの DATA ステートメントに指定したデータセット数と同じ数までデータセットを指定できます。

---

## 詳細

### OUTPUT ステートメントがオブザベーションを書き込むタイミングと場所

OUTPUT ステートメントは、DATA ステップの最後ではなく、現在のオブザベーションをすぐに SAS データセットに書き込みます。OUTPUT ステートメントにデータセット名が指定されていない場合、DATA ステートメントにリストされている 1 つまたは複数のデータセットにオブザベーションが書き込まれます。

### 暗示的な OUTPUT と明示的な OUTPUT

デフォルトでは、すべてのデータステップで繰り返しの最後に暗示的な OUTPUT ステートメントが含まれ、作成中の 1 つまたは複数のデータセットにオブザベーションが書き込まれます。OUTPUT ステートメントを DATA ステップに明示的に指定すると、この自動出力より優先されるため、明示的な OUTPUT ステートメントの実行時のみオブザベーションがデータセットに追加されるようになります。OUTPUT ステートメントを使用して 1 つでもデータセットにオブザベーションを書き込むようにすると、DATA ステップの最後に暗示的な OUTPUT ステートメントは存在しなくなります。そのため、DATA ステップでオブザベーションが書き込まれるのは、明示的な OUTPUT ステートメントの実行時のみになります。OUTPUT ステートメントはそれだけでも使用できますが、IF-THEN ステートメント、SELECT ステートメント、DO ループの処理と組み合わせても使用できます。

### MODIFY ステートメントを使用する場合

MODIFY ステートメントを OUTPUT ステートメントと組み合わせて使用する場合、REMOVE および REPLACE の各ステートメントは、DATA ステップの繰り返しの最後の暗示的な書き込みアクションより優先されます。詳細については、“[比較](#)” (315 ページ) を参照してください。特定のオブザベーションに対して OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を実行する場合、オブザベーションのポインタの位置を正しく保つため、OUTPUT ステートメントを最後に実行します。

## 比較

- OUTPUT ステートメントはオブザベーションを SAS データセットに書き込みますが、PUT ステートメントは変数の値またはテキスト文字列を外部ファイルや SAS ログに書き込みます。
- 指定した出力データセットにオブザベーションを書き込むタイミングを制御するには、OUTPUT ステートメントを使用します。指定した出力データセットに書き込む変数を制御するには、DATA ステートメントで KEEP= または DROP= データセットオプションを使用するか、KEEP または DROP ステートメントを使用します。
- MODIFY ステートメントと組み合わせて OUTPUT ステートメントを使用する場合、次の規則が適用されます。
  - OUTPUT、REPLACE、REMOVE の各ステートメントの使用は、DATA ステップの最後に実行されるデフォルトの書き込みアクションより優先されます。(OUTPUT がデフォルトのアクションです。ただし、MODIFY ステートメントを使用すると REPLACE がデフォルトのアクションになります。)これらのステートメントのいずれかを DATA ステップで使用する場合、データセットに新しいオブザベーションが追加されるように出力を明示的にプログラムする必要があります。

- OUTPUT、REPLACE、REMOVE の各ステートメントは互いに独立して動作します。順序が論理的に正しければ、同じオブザベーションに複数のステートメントを適用できます。
- 特定のオブザベーションに対して OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を実行する場合、オブザベーションのポインタの位置を正しく保つため、OUTPUT ステートメントを最後に実行します。

## 例

### 例 1: OUTPUT ステートメントの使用例

これらの例では、OUTPUT ステートメントの使用方法を示します。

- 次のコード行では、現在のオブザベーションが SAS データセットに書き込まれます。

```
output;
```

- 次のコード行では、指定した条件が真の場合に現在のオブザベーションが SAS データセットに書き込まれます。

```
if deptcode gt 2000 then output;
```

- 次のコード行では、PHONE の値が存在しない場合にオブザベーションがデータセット MARKUP に書き込まれます。

```
if phone=. then output markup;
```

### 例 2: 1つの入力行から複数のオブザベーションを作成する

入力データの 1 つの行から複数のオブザベーションを作成することができます。次の SAS プログラムでは、データセット SULFA 内の 1 つのオブザベーションから、データセット RESPONSE 内に 3 つのオブザベーションが作成されます。

```
data response(drop=time1-time3);
  set sulfa;
  time=time1;
  output;
  time=time2;
  output;
  time=time3;
  output;
run;
```

### 例 3: 1つの入力ファイルから複数のデータセットを作成する

1 つの入力ファイルから複数の SAS データセットを作成できます。この例の OUTPUT ステートメントは、2 つのデータセット OZONE と OXIDES にオブザベーションを書き込みます。

```
data ozone oxides;
  infile file-specification;
  input city $ 1-15 date date9.
        chemical $ 26-27 ppm 29-30;
  if chemical='O3' then output ozone;
  else output oxides;
run;
```



**例 4: 複数の入力行から 1 つのオブザベーションを作成する**

複数の入力オブザベーションを 1 つのオブザベーションにまとめることができます。この例の OUTPUT ステートメントは、入力データセットにある最初の 10 個のオブザベーションの DEFECTS 値を合計し、1 つのオブザベーションを作成します。

```
data discards;
  set gadgets;
  drop defects;
  reps+1;
  if reps=1 then total=0;
  total+defects;
  if reps=10 then do;
    output;
    stop;
  end;
end;
run;
```

**関連項目:****ステートメント:**

- [“DATA ステートメント” \(47 ページ\)](#)
- [“MODIFY ステートメント” \(289 ページ\)](#)
- [“PUT ステートメント” \(318 ページ\)](#)
- [“REMOVE ステートメント” \(354 ページ\)](#)
- [“REPLACE ステートメント” \(357 ページ\)](#)

---

**PAGE ステートメント**

SAS ログの新しいページにスキップします。

**該当要素:** 任意の場所

**カテゴリ:** ログ制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

**構文**

**PAGE;**

**引数なし**

PAGE ステートメントを使用すると、SAS ログの新しいページにスキップします。

**詳細**

PAGE ステートメントはバッチモード、非対話型モードで SAS を実行する場合に使用できます。PAGE ステートメント自体はログに表示されません。対話型ラインモードで SAS を実行する場合、PAGE ステートメントを使用すると画面モニタ(または代替ログファイル)に空白行が出力されることがあります。

**関連項目:****ステートメント:**

- [“LIST ステートメント” \(268 ページ\)](#)

**システムオプション:**

- [“LINESIZE= System Option” \(SAS Viya System Options: Reference\)](#)
- [“PAGESIZE= System Option” \(SAS Viya System Options: Reference\)](#)

---

**PUT ステートメント**

SAS ログ、結果ウィンドウ、または最後の FILE ステートメントに指定した外部の場所に行を出力します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 実行

---

**構文**

**PUT** <specification(s)> <\_ODS\_> <@ | @@>;

**引数なし**

引数を指定しない PUT ステートメントは、ヌル PUT ステートメントと呼ばれます。ヌル PUT ステートメントは、次のように動作します。

- 現在の出力行がブランクの場合でも、現在の出力行を現在の場所へ書き込みます。
- 前回の PUT ステートメントの後置@で保持された出力行を解除します。

例については、“[例 5: 出力行の保持と解放](#)” (333 ページ)を参照してください。詳細については、“[ラインホールド指定子の使用](#)” (327 ページ)を参照してください。

**引数****specification(s)**

書き込む内容、書き込み方法、書き込み先を指定します。specification には次を指定できます。

**variable**

変数の名前を指定します。指定した変数の値が書き込まれます。

注: バージョン 7 から、列をマップした ODS(Output Delivery System)変数を PUT ステートメントに指定できます。この機能の概要については、[\\_ODS\\_](#) (320 ページ)で説明されています。

**(variable-list)**

変数のリストを指定します。指定した変数の値が書き込まれます。

要件 [\(format-list\)](#)は[\(variable-list\)](#)の後ろに指定する必要があります。

---

参照項目: [“PUT ステートメント、フォーマット” \(338 ページ\)](#)

### ***'character-string'***

書き込む文字列を一重引用符で囲んで指定します。

ヒント 16 進数の文字列を EBCDIC または ASCII で書き込むには、終了引用符の後ろに **x** を追加します。

間に空白を含まない一重引用符(')または二重引用符("")の記号のみをテキスト文字列として指定すると、一重引用符(')または二重引用符("")がそれぞれ生成されます。

参照項目: [“リスト出力” \(325 ページ\)](#)

例 次のステートメントでは、16 進数の文字列を ASCII 文字に変換してから、HELLO と書き込みます。  

```
put '68656C6C6F';
```

### ***n\****

後ろに指定する文字列を *n* 回繰り返します。

例 次のステートメントでは、出力行にアンダースコアを 132 回書き込みます。  

```
put 132*'_';
```

例 [“例 4: テキストに下線を引く” \(333 ページ\)](#)

### ***pointer-control***

出力バッファ内の指定した行または列に出力ポインタを移動させます。

参照項目: [“カラムポインタコントロール” \(321 ページ\)](#)  
[“行ポインタコントロール” \(322 ページ\)](#)

### ***column-specifications***

値を書き込む出力行内の列位置を指定します。

参照項目: [“カラム出力” \(324 ページ\)](#)

例 [“例 2: ポインタをページ内で移動させる” \(330 ページ\)](#)

### ***format.***

変数の値を書き込むときに使用する出力形式を指定します。

参照項目: [“フォーマット出力” \(325 ページ\)](#)

例 [“例 1: 1 つの PUT ステートメントで複数の出力スタイルを使用する” \(329 ページ\)](#)

### ***(format-list)***

出力形式のリストを指定します。このリストは、前に指定した変数リストの値を書き込むときに使用します。

制限事項 *(format-list)*は*(variable-list)*の後ろに指定する必要があります。

参照項目: [“PUT ステートメント、フォーマット” \(338 ページ\)](#)

**\_INFILE\_**

現在の入力ファイルから、または DATELINES ステートメントの後ろに指定したデータ行のどちらかから読み込んだ最後の入力データレコードを書き込みます。

**ヒ** \_INFILE\_ は、現在の INPUT バッファを参照する自動変数です。他の SAS ステートメントでもこの自動変数を使用できます。

最後に使用した INPUT ステートメントが行ポインタコントロールを使用して複数の入力データレコードを読み込む場合、PUT \_INFILE\_ では、入力ポインタが置かれたレコードのみを書き込みます。

**例** 次の PUT ステートメントでは、1 行目の入力データレコードの値をすべて書き込みます。

```
input #3 score #1 name $ 6-23;
put _infile_;
```

**例** “例 6: 現在の入力レコードをログに書き込む” (334 ページ)

**\_ALL\_**

現在の DATA ステップにて定義されるすべての変数の値を、自動変数を含めて書き込みます。名前付き出力を使用します。

参照項目: “名前付き出力” (325 ページ)

**\_ODS\_**

(FILE ステートメントの ODS オプションで定義される)すべての列のデータ値を特別なバッファに移動します。データの値は移動先からデータコンポーネントに書き込まれます。FILE ステートメントの ODS オプションは、DATA ステップの結果を保持するデータコンポーネントの構造を定義します。

**制限事項** SAS Viya では、ODS 出力時に PUT ステートメントは VARCHAR 変数をサポートしていません。

\_ODS\_ は、FILE ステートメントに ODS オプションを事前に指定した場合にのみ使用できます。

**操作** \_ODS\_ による特定の列へのデータの書き込みは、PUT ステートメントでカラムポインタを使用して、その列に変数を指定していない場合に限られます。つまり、列への変数の指定は、\_ODS\_ オプションより優先されます。

**ヒント** 変数の指定やカラムポインタと組み合わせて、\_ODS\_ を指定できます。また、PUT ステートメントのどの位置にでも指定できます。

**@|@@**

次の PUT ステートメントの実行時に使用できるように出力行を保持します。このラインホールド指定子は、後置@および後置@@と呼ばれます。

**制限事項** 後置@または後置@@は、PUT ステートメントの最後の項目として指定する必要があります。

**ヒント** ポインタを現在の位置に保持するには、@または@@を使用します。次の PUT ステートメントは、新しい出力行ではなく同じ出力行に書き込みます。

参照項目:	<a href="#">“ラインホールド指定子の使用” (327 ページ)</a>
例	<a href="#">“例 5: 出力行の保持と解放” (333 ページ)</a>

## カラムポインタコントロール

### @*n*

ポインタを列 *n* に移動させます。

範囲 正の整数

例 正の整数@15 が指定されているため、NAME の値を書き込む前に、ポインタを列 15 に移動させます。  

```
put @15 name $10.;
```

例 [“例 2: ポインタをページ内で移動させる” \(330 ページ\)](#)および  
[“例 4: テキストに下線を引く” \(333 ページ\)](#)

### @*numeric-variable*

指定した *numeric-variable* の値が示す列にポインタを移動させます。

範囲 正の整数

ヒント *n* が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*n* が 0 または負の場合、ポインタは列 1 に移動します。

例 変数 A の値に従って、NAME 変数の値を書き込む前に、ポインタを列 15 に移動させます。  

```
a=15;
put @a name $10.;
```

例 [“例 2: ポインタをページ内で移動させる” \(330 ページ\)](#)

### @(*expression*)

指定した *expression* の値が示す列にポインタを移動させます。

範囲 正の整数

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。値が 0 になる場合、ポインタは列 1 に移動します。

例 式の結果に従って、NAME 変数の値を書き込む前に、ポインタを列 15 に移動させます。  

```
b=5;
put @(b*3) name $10.;
```

### +*n*

*n* に指定した列数だけポインタを移動させます。

範囲 正の整数または 0

ヒント *n* が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

例 次のステートメントでは、ポインタを列 23 に移動させ、LENGTH の値を列 23 から 26 に書き込みます。次に、ポインタを 5 列だけ右に移動させ、列 32 から 35 に WIDTH の値を書き込みます。

```
put @23 length 4. +5 width 4.;
```

**+numeric-variable**

指定した *numeric-variable* の値が示す列数だけポインタを移動させます。

範囲 正の整数、負の整数、または 0

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*numeric-variable* の値が負の値の場合、ポインタを後方(左)へ移動させます。ただし、現在の列位置が 1 未満になる場合、ポインタは列 1 に移動します。値が 0 の場合、ポインタは移動しません。この値が出力バッファの長さを超える場合、現在の行を書き込んだ後、ポインタは次の行の列 1 に移動します。

**+(expression)**

指定した *expression* の値が示す列数だけポインタを移動させます。

範囲 *expression* の実行結果は正の整数でなければなりません。

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*expression* の実行結果が負の値の場合、ポインタを左に戻します。ただし、現在の列位置が 1 未満になる場合、ポインタは列 1 に移動します。値が 0 の場合、ポインタは移動しません。この値が出力バッファの長さを超える場合、現在の行を書き込んだ後、ポインタは次の行の列 1 に移動します。

例 “例 2: ポインタをページ内で移動させる” (330 ページ)

**行ポインタコントロール****#n**

行 *n* の列 1 にポインタを移動させます。

範囲 正の整数

例 正の整数 #2 と指定されているので、ポインタは ID の値を列 3 と 4 に書き込む前に 2 行目に移動します。

```
put @12 name $10. #2 id 3-4;
```

**#numeric-variable**

指定した *numeric-variable* の値が示す行の列 1 にポインタを移動させます。

範囲 正の整数

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

 **#(expression)**

指定した *expression* の値が示す行の列 1 にポインタを移動させます。

範囲 *expression* の実行結果は正の整数でなければなりません。

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

/

ポインタを次の行の列 1 に移動させます。

注 1 つまたは複数の"/"行ポインタコントロールを使用して SAS ログにブランク行を追加しようとしても、ブランク行は生成されません。それ以外の出力形式では、ブランク行が生成されます。

例 変数 NAME と AGE の値を 1 行目に書き込んでから、ポインタを 2 行目に移動させ、列 3 と 4 に変数 ID の値を書き込みます。

```
put name age / id 3-4;
```

例 “例 3: ポインタを新しいページに移動させる” (332 ページ)

### OVERPRINT

キーワード OVERPRINT の後ろに指定した値を、直前に書き込んだ出力行に重ねて書き込みます。

要件 この出力はファイルに送信する必要があります。FILE ステートメントの N=オプションを 1 に設定し、PUT ステートメントの出力先をファイルに設定してください。

ヒント OVERPRINT は、画面に書き込まれる行には影響しません。

テキストを重ねて出力する場合、カラムポインタコントロールと行ポインタコントロールと組み合わせて OVERPRINT を使用します。

例 次のステートメントでは列 15 からアンダースコアを重ねて出力し、タイトルに下線を表示します。

```
put @15 'Report Title' overprint
    @15 '_____';
```

例 “例 4: テキストに下線を引く” (333 ページ)

### \_BLANKPAGE\_

ポインタを新しいページの最初の行に移動させます。ポインタが新しいページの最初の行の最初の列にある場合でも移動を実行します。

ヒント 現在の出力ファイルにキャリッジコントロール文字が含まれている場合、\_BLANKPAGE\_は適切なキャリッジコントロール文字を含む出力行を生成します。

例 “例 3: ポインタを新しいページに移動させる” (332 ページ)

### \_PAGE\_

ポインタを新しいページの最初の行に移動させます。行が現在の PAGESIZE=に指定された値を超えると、自動的に新しいページが開始されず。

ヒント 現在の出力ファイルを書き込む場合に、\_PAGE\_オプションは適切なキャリッジコントロール文字を含む出力行を生成します。\_PAGE\_オプションは、書き込まれていないファイルには影響しません。

対話型 SAS セッションで FILE PRINT を指定すると、**結果**ウィンドウではフォームフィードコントロール文字が改ページと解釈されるので、その文字は出力から削除されます。結果として生成されるファイルは、改ページ文字を含まないフラットファイルになります。フォームフィード文字をファイルに含める必要がある場合、FILE ステートメントに物理ファイルの場所と PRINT オプションを指定する必要があります。

例 “例 3: ポインタを新しいページに移動させる” (332 ページ)

## 詳細

### PUT ステートメントを使用する必要がある場合

SAS ログ、**結果**ウィンドウ、または外部の場所に行を出力する場合は、PUT ステートメントを使用します。DATA ステップのの反復時、PUT ステートメントの前に FILE ステートメントを実行しない場合、行は SAS ログに書き込まれます。FILE ステートメントに PRINT オプションを指定すると、行は**結果**ウィンドウに書き込まれます。

PUT ステートメントでは、変数の値、文字列、16 進数の文字定数を含む行を書き込むことができます。PUT ステートメントの指定を使用して、書き込み対象、書き込み先、および書き込み時の出力形式を指定できます。

## 出力スタイル

### 出力スタイルの概要

PUT ステートメントを使用して変数の値を書き込む場合、次の 4 つの方法を使用します。

- カラム
- リスト(単純および修飾)
- フォーマット
- 名前付き

1 つの PUT ステートメントに、行の出力に合わせて、利用可能な出力スタイルの一部またはすべてを使用できます。

### カラム出力

*カラム出力*を使用する場合、PUT ステートメントで、値の位置を示す列番号を変数の後ろに指定します。番号は、値を書き込む行内での位置を示しています。

```
put name 6-15 age 17-19;
```

次の行が SAS ログに書き込まれます。

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

```
----+----1-----2----+
      Peterson  21
      Morgan    17
```

PUT ステートメントによって、変数 NAME と AGE の値が指定した列に書き込まれます。詳細については、“PUT ステートメント、カラム”(336 ページ)を参照してください。



### リスト出力

*リスト出力*を使用する場合、書き込む順序で PUT ステートメントに変数と文字列を指定します。例えば、この PUT ステートメントによって、NAME と AGE の値は SAS ログに書き込まれます。

```
put name age;
```

次に SAS ログを示します。

```
----+----1----+----2----+
Peterson 21
Morgan 17
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

詳細については、“[PUT ステートメント、リスト](#)” (343 ページ)を参照してください。

### フォーマット出力

*フォーマット出力*を使用する場合、変数名の後ろに SAS 出力形式またはユーザー定義の出力形式を指定します。この出力形式によって、変数の値を書き込む方法が指示されます。出力形式を指定すると、パック 10 進などの非標準形式のデータや、カンマなどの特殊文字を含む数値を書き込むことができます。-L、-C、-R を使用すると、フォーマット出力のデフォルトの配置より優先されます。

例えば、この PUT ステートメントによって、NAME、AGE、および DATE の値は SAS ログに書き込まれます。

```
put name $char10. age 2. +1 date mmddy10.;
```

次に SAS ログを示します。

```
----+----1----+----2----+
Peterson 21 07/18/1999
Morgan 17 11/12/1999
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

ポインタコントロールを+1 と指定すると、変数 AGE と DATE の値の間に空白が 1 つ挿入されます。詳細については、“[PUT ステートメント、フォーマット](#)” (338 ページ)を参照してください。

### 名前付き出力

*名前付き出力*を使用する場合、変数名の前に等号を付けてリストします。例えば、この PUT ステートメントによって、NAME と AGE の値は SAS ログに書き込まれます。

```
put name= age=;
```

次に SAS ログを示します。

```
----+----1----+----2----+
name=Peterson age=21
name=Morgan age=17
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

詳細については、“[PUT ステートメント、名前付き](#)” (347 ページ)を参照してください。

**1 つの PUT ステートメントで複数の出力スタイルを使用する**

1 つの PUT ステートメントに、異なる出力スタイルの一部またはすべてを組み合わせることができます。次に例を示します。

```
put name 'on ' date mmdyy8. ' weighs '
    startwght +(-1) ' idno= 40-45;
```

SAS ログへの行の書き込みについては、“[例 1: 1 つの PUT ステートメントで複数の出力スタイルを使用する](#)” (329 ページ)を参照してください。

異なる出力スタイルを組み合わせる場合、それぞれの値を書き込んだ後の出力ポインタの位置を理解することが重要です。ポインタの位置の詳細については、“[値を書き込んだ後のポインタの位置](#)” (328 ページ)を参照してください。

**文字定数と変数の書き込み時によく発生するエラーを回避する**

PUT ステートメントを使用して文字定数の後ろに変数名を書き込む場合、終了引用符と変数名の間に常に空白を 1 つ追加します。

```
put 'Player:' name1 'Player:' name2 'Player:' name3;
```

空白を追加しないと、次の表に示すように、変数名が後ろに続く文字定数は特殊な SAS 定数と解釈される場合があります。

表 2.9 文字定数の後ろに追加すると正しく解釈されない文字

変数の開始文字	説明	例
b	ビットテスト定数	'00100000'b
d	日付定数	'01jan04'd
dt	日時定数	'18jan2003:9:27:05am'dt
n	名前リテラル	'My Table'n
t	時間定数	'9:25:19pm't
x	16 進法表記	'534153'x

“[例 7: 変数が後ろに続く文字定数の書き込み時によく発生するエラーを回避する](#)” (335 ページ)では、文字定数の後ろに変数を使用する方法について説明しています。

**ポインタコントロール**

PUT ステートメントの値を書き込むときは、ポインタを使用して書き込み位置を制御します。PUT ステートメントには、ポインタの動きを制御する 3 つの方法があります。

**カラムポインタコントロール**

PUT ステートメントで出力行への値の書き込みを開始するときに、ポインタの列の位置をリセットします。

**行ポインタコントロール**

PUT ステートメントで出力行に書き込むときに、ポインタの行の位置をリセットします。

### ラインホールド指定子

出力バッファに行が保持されるので、他の PUT ステートメントでも同じ行に書き込むことができます。デフォルトでは、PUT ステートメントを実行すると、以前の行を解放して、新しい行に書き込みます。

カラムポインタコントロールや行ポインタコントロールを使用すると、ポインタを移動させる行または列の絶対位置を指定したり、現在のポインタの位置の相対位置として移動先の列または行を指定できます。PUT ステートメントで指定できるすべてのポインタコントロールを次の表に示します。

表 2.10 PUT ステートメントで指定できるポインタコントロール

ポインタコントロール	相対指定	絶対指定
カラムポインタコントロール	$+n$	@ $n$
	$+numeric-variable$	@ $numeric-variable$
	$+(expression)$	@( $expression$ )
行ポインタコントロール	$/$ , <code>_PAGE_</code> , <code>_BLANKPAGE_</code>	# $n$ # $numeric-variable$ #( $expression$ )
	OVERPRINT	なし
	ラインホールド指定子	@
	@@	(なし)

注: ポインタコントロールは、常に適用する変数の前に指定するようにしてください。

SAS でポインタ位置を特定する方法については、“[値を書き込んだ後のポインタの位置](#)” (328 ページ) を参照してください。

### ラインホールド指定子の使用

ラインホールド指定子を使用して、次の場合にポインタを現在の出力行に保持します。

- 複数の PUT ステートメントを使用して値を同じ出力行に書き込む場合
- 1 つの PUT ステートメントを使用して複数のオブザベーションの値を同じ出力行に書き込む場合

ラインホールド指定子を指定しない場合、DATA ステップの PUT ステートメントごとに値が新しい出力行に書き込まれます。

PUT ステートメントでは、後置@と後置@@は同じ働きをします。INPUT ステートメントとは異なり、PUT ステートメントでは、DATA ステップの新しい繰り返しの開始時に、後置@で保持されている行は自動的に解放されません。後置@または後置@@で保持されている現在の出力行は、次の場合に解放されます。

- PUT ステートメントに後置@が指定されていない場合

- PUT ステートメントに `_BLANKPAGE_` または `_PAGE_` が指定されている場合
- 現在の行の行端に達した場合 (FILE ステートメントの `LRECL=` オプションまたは `LINESIZE=` オプションの現在の値、または指定されている場合には `LINESIZE=` システムオプションの現在の値により決定)
- DATA ステップで最後の繰り返しが終了した場合

後置@または後置@@を使用すると、次の PUT ステートメントの実行時にポインタの値が変更されないため、現在の行の長さを超えても値を書き込もうとする場合があります。“[ポインタが行の終わりを超える場合](#)” (328 ページ) を参照してください。

### 値を書き込んだ後のポインタの位置

1 つの PUT ステートメントで複数の出力スタイルを組み合わせる場合は、値を書き込んだ後の出力ポインタの位置を理解しておくことが重要になります。値を書き込んだ後のポインタの位置は、指定した出力スタイルや、書き込んだ値が文字列か変数かによって異なります。カラム出力またはフォーマット出力を使用すると、ポインタの位置は PUT ステートメントに指定したフィールドの最後から 1 番目の列になります。この 2 つのスタイルでは、変数の値のみが書き込まれます。

リスト出力と名前付き出力を使用すると、PUT ステートメントでは、それぞれの値を書き込んだ後に列を 1 つ自動的にスキップするので、ポインタの位置は変数の値の最後から 2 番目の列になります。ただし、PUT ステートメントでリスト出力を使用して文字列を書き込む場合、ポインタの位置は文字列の最後から 1 番目の列になります。文字列を書き込んだ後に行ポインタコントロールやカラム出力を使用しない場合、文字列と次の値を区切るため、文字列の最後に空白を 1 つ追加してください。

`_INFILE_` の指定時、ポインタは現在の入力ファイルのレコードを書き込んだ後に列 1 に配置されます。

出力ポインタは、次の場合にページの上部左端に表示されます。

- `PUT _BLANKPAGE_` が空白ページを書き込み、ポインタを次のページの先頭に移動させた場合
- `PUT _PAGE_` が同じ位置にポインタを保持する場合

FILE ステートメントの `COLUMN=` オプションまたは `LINE=` オプションに指定した変数の値を調べて、現在のポインタの位置を特定できます。

### ポインタが行の終わりを超える場合

現在指定されている出力行の長さを超える場合、出力行は書き込まれません。現在の出力行の長さは次の値で決定されます。

- 現在の FILE ステートメントの `LINESIZE=` オプションの値
- `LINESIZE=` システムオプションの値 (結果 ウィンドウの場合)
- 現在の FILE ステートメントの `LRECL=` オプションの値 (外部ファイルの場合)

次に示す指定を 1 つまたは複数使用すると、ポインタの位置が誤って現在の行の長さを超えてしまう場合があります。

- 現在の行の長さを超えた列にポインタを移動させるように +ポインタコントロールの値が指定されている場合
- 列範囲が現在の行の長さを超えている場合 (たとえば、現在の行の長さが 80 の場合に `PUT X 90 - 100` と指定する場合)
- 変数の値または文字列が現在の出力行のスペースよりも長い場合

デフォルトでは、PUT ステートメントが現在の行の終わりを越えて書き込む必要がある場合、現在の行に収まらない項目の書き込みは行わずに、現在の行の書き込みを終了します。次に、現在の行に書き込めなかった項目を新しい行の列 1 から書き込みます。“FILE ステートメント” (75 ページ) の FLOWOVER オプション、DROPOVER オプション、STOPOVER オプションを参照してください。

## 配列

PUT ステートメントを使用して配列要素を書き込むことができます。subscript には、PUT ステートメントの実行時に結果が整数になる SAS 式を指定できます。次に示すように、配列参照を丸かっこで囲むと、ポインタコントロール付きの *numeric-variable* の構文に配列参照を使用できます。

- `@(array-name{i})`
- `+(array-name{i})`
- `#(array-name{i})`

事前に定義した配列のすべての要素を外部の場所へ書き込むには、配列の subscript にアスタリスク(\*)を使用します。1 次元配列または多次元配列は使用できますが、\_TEMPORARY\_配列は使用できません。subscript を中かっこ、大かっこ、丸かっこで囲みます。次に、リスト出力、フォーマット出力、カラム出力、名前付き出力を使用して配列を書き込みます。リスト出力を使用する場合、ステートメントの形式は次のようになります。

```
PUT array-name{*};
```

フォーマット出力を使用する場合、ステートメントの形式は次のようになります。

```
PUT array-name{*}(format|format.list)
```

丸かっこで囲んだ出力形式は、配列参照の後ろに続けて指定します。

## 比較

- PUT ステートメントでは変数の値や文字列を SAS ログや外部の場所へ書き込みますが、INPUT ステートメントでは外部ファイルの生データやインストリームで入力されるデータ行を読み込みます。
- INPUT ステートメントと PUT ステートメントのどちらも、後置@や後置@@ ラインホールド指定子を使用すると、入力バッファや出力バッファ内に現在の行が保持されます。INPUT ステートメントでは、後置@@を使用して入力バッファに行を保持し、DATA ステップの繰り返し時にそのまま使用します。PUT ステートメントでは、後置@は後置@@と同じ動作になります。どちらを使用しても、DATA ステップの反復間を通して行を保持します。
- PUT ステートメントと OUTPUT ステートメントはどちらも DATA ステップで出力を生成します。PUT ステートメントは出力バッファを使用し、外部の場所、SAS ログ、またはモニタに出力を書き込みます。OUTPUT ステートメントは、プログラムデータベクトルを使用し、SAS データセットにオブザーベーションを書き込みます。

## 例

### 例 1: 1 つの PUT ステートメントで複数の出力スタイルを使用する

この例では、1 つの PUT ステートメントで複数の出力スタイルを使用します。

```
data club1;
  input idno name $ startwght date : date7.;
```

```

put name 'on ' date mmdyy8. ' weighs '
    startwght +(-1) ' ' idno= 32-40;
datalines;
032 David 180 25nov99
049 Amelia 145 25nov99
219 Alan 210 12nov99
;

```

この例の各変数に指定されている出力スタイルを次の表に示します。

変数	出力スタイル
NAME, STARTWGHT	リスト出力
DATE	フォーマット出力
IDNO	名前付き出力

この PUT ステートメントではポインタコントロールも使用しています。また、文字列と変数名も指定しています。

このプログラムを実行すると、次の行が SAS ログに出力されます。

```

----+----1----+----2----+----3----+----4
David on 11/25/99 weighs 180. idno=1032
Amelia on 11/25/99 weighs 145. idno=1049
Alan on 11/12/99 weighs 210. idno=1219

```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

文字列の先頭と末尾に空白が挿入されているので、ポインタの位置が移動します。この空白によって、変数の値と文字列が区切られています。ポインタコントロールを+(-1)と指定しているため、ポインタが後方(左)に移動し、STARTWGHT の値とピリオドの間の不要な空白が削除されます。ポインタ位置を指定する方法の詳細については、“[値を書き込んだ後のポインタの位置](#)” (328 ページ)を参照してください。

## 例 2: ポインタをページ内で移動させる

次の PUT ステートメントは、カラムポインタコントロールと行ポインタコントロールを使用して出力ポインタの位置を指定する方法を示しています。

- ポインタを特定の列に移動させるには、@の後ろに列番号か、列番号の値を示す変数や式を指定します。たとえば、次のステートメントではポインタを列 15 に移動してから、リスト出力を使用して変数 TOTAL SALES の値を書き込みます。

```
put @15 totalsales;
```

次の PUT ステートメントでは、ポインタを変数 COLUMN に指定した値に移動した後、COMMA6 出力形式を使用して変数 OTAL SALES の値を書き込みます。

```

data _null_;
  set carsales;
  column=15;
  put @column totalsales comma6.;
run;

```

- 次のプログラムでは、ポインタを戻す 2 つの方法を示しています。

```
data carsales;
  input item $10. jan : comma5.
        feb : comma5. mar : comma5.;
  saleqtr1=sum(jan,feb,mar);
/* an expression moves pointer backward */
  put '1st qtr sales for ' item
     'is ' saleqtr1 : comma6. +(-1) '!;
/* a numeric variable with a negative
   value moves pointer backward. */
  x=-1;
  put '1st qtr sales for ' item
     'is ' saleqtr1 : comma5. +x '!;
  datalines;
trucks   1,382   2,789   3,556
vans     1,265   2,543   3,987
sedans   2,391   3,011   3,658
;
```

SALEQTR1 の値は修飾リスト出力を使用して書き込まれるので、ポインタは自動的に 2 列先に移動します。詳細については、“[修飾リスト出力とフォーマット出力の違い](#)” (345 ページ) を参照してください。値とピリオドの間の不要な空白を削除するために、ポインタを 1 列だけ戻します。

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
----+----1----+----2----+----3----+----4
st qtr sales for trucks is 7,727.
st qtr sales for trucks is 7,727.
st qtr sales for vans is 7,795.
st qtr sales for vans is 7,795.
st qtr sales for sedans is 9,060.
st qtr sales for sedans is 9,060.
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

- 次のプログラムでは、PUT ステートメントと行ポインタコントロール(/)を使用して、次行にポインタを移動させます。

```
data _null_;
  set carsales end=lastrec;
  totalsales+saleqtr1;
  if lastrec then
    put @2 'Total Sales for 1st Qtr'
       / totalsales 10-15;
run;
```

この DATA ステップでは、CARSALES データセット内のすべてのオブザベーションを使用して TOTALSALES の値を算出した後、PUT ステートメントを実行します。列 2 から文字列を書き込み、次の行に移動してから列 10 から 15 に TOTALSALES の値を書き込みます。

```
----+----1----+----2----+----3
Total Sales for 1st Qtr
      24582
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。



**例 3: ポインタを新しいページに移動させる**

この例では STATEPOP というデータセットを作成します。このデータセットには、1990 年の米国国勢調査の都市圏と非都市圏の人口に関する情報が含まれています。ここでは、FORMAT プロシジャを実行し、50 州とコロンビア特別区を 4 つの地域にグループ化しています。次に、IF ステートメントと PUT ステートメントを使用して出力を制御しています。

```

title1;
data statepop;
  input state $ cityp90 ncityp90 region @@;
  label cityp90='1990 metropolitan population
          (million)'
        ncityp90='1990 nonmetropolitan population
          (million)'
        region='Geographic region';
  datalines;
ME .443 .785 1 NH .659 .450 1
VT .152 .411 1 MA 5.788 .229 1
RI .938 .065 1 CT 3.148 .140 1
NY 16.515 1.475 1 NJ 7.730 .A 1
PA 10.083 1.799 1 DE .553 .113 2
MD 4.439 .343 2 DC .607 . 2
VA 4.773 1.414 2 WV .748 1.045 2
NC 4.376 2.253 2 SC 2.423 1.064 2
GA 4.352 2.127 2 FL 12.023 .915 2
KY 1.780 1.906 2 TN 3.298 1.579 2
AL 2.710 1.331 2 MS .776 1.798 2
AR 1.040 1.311 2 LA 3.160 1.060 2
OK 1.870 1.276 2 TX 14.166 2.821 2
OH 8.826 2.021 3 IN 3.962 1.582 3
IL 9.574 1.857 3 MI 7.698 1.598 3
WI 3.331 1.561 3 MN 3.011 1.364 3
IA 1.200 1.577 3 MO 3.491 1.626 3
ND .257 .381 3 SD .221 .475 3
NE .787 .791 3 KS 1.333 1.145 3
MT .191 .608 4 ID .296 .711 4
WY .134 .319 4 CO 2.686 .608 4
NM .842 .673 4 AZ 3.106 .559 4
UT 1.336 .387 4 NV 1.014 .183 4
WA 4.036 .830 4 OR 1.985 .858 4
CA 28.799 .961 4 AK .226 .324 4
HI .836 .272 4
;
proc format;
  value regfmt 1='Northeast'
              2='South'
              3='Midwest'
              4='West';
run;
data _null_;
  set statepop;
  by region;
  pop90=sum(cityp90,ncityp90);
  file print;
  put state 1-2 @5 pop90 7.3 ' million';
  if first.region then

```



```

    regioncitypop=0; /* new region */
regioncitypop+cityp90;
if last.region then
do;
    put // '1990 US CENSUS for ' region regfmt.
        / 'Total Urban Population:'
            regioncitypop' million' _page_;
end;
run;

```

#### アウトプット 2.26 PUT ステートメントを使用した北東地域の出力結果

```

1 ME 1.228 million NH 1.109 million VT 0.563 million MA 6.017 million RI 1.003
million CT 3.288 million NY 17.990 million NJ 7.730 million PA 11.882 million
1990 US CENSUS for Northeast Total Urban Population: 45.456 million

```

LAST.REGION の値が 1 の場合、PUT \_PAGE\_ はポインタを新しいページの 1 行に移動させます。例では、ページから移動の前にフッターメッセージが表示されます。

#### 例 4: テキストに下線を引く

この例では、OVERPRINT を使用し、直前の PUT ステートメントで書き込んだ値の下に下線を表示します。

```

data _null_;
input idno name $ startwght;
file file-specification print;
put name 1-10 @15 startwght 3.;
if startwght > 200 then
    put overprint @15 '___';
datalines;
032 David 180
049 Amelia 145
219 Alan 210
;

```

2 番目の PUT ステートメントでは、最初の PUT ステートメントで書き込んだ体重の値が 200 ポンドを超えていた場合に下線を引きます。

次の PUT ステートメントでは、カラムポインタコントロールと行ポインタコントロールとともに OVERPRINT を使用しています。

```

put @5 name $8. overprint @5 8*'_'
  / @20 address;

```

この PUT ステートメントでは、変数 NAME の値を書き込んでから、8 個のアンダースコアを重ねて下線を引きます。その後、出力ポインタを次の行に移動させ、変数 ADDRESS の値を書き込みます。

#### 例 5: 出力行の保持と解放

次の DATA ステップでは、PUT ステートメントで出力行の保持と解放を行う方法を示します。

```

data _null_;
input idno name $ startwght 3.;
put name @;
if startwght ne . then
    put @15 startwght;

```

```

else put;
datalines;
032 David 180
049 Amelia 145
126 Monica
219 Alan 210
;

```

この例では、

- 最初の PUT ステートメントに後置@を指定し、変数 NAME の値を書き込んだ後、現在の出力行を保持します。
- IF-THEN ステートメントの条件が真になる場合、2 番目の PUT ステートメントで STARTWGHT の値を書き込んでから、現在の出力行を解放します。
- 条件が偽になる場合、2 番目の PUT ステートメントは実行されません。かわりに、ELSE PUT ステートメントが実行されます。ELSE PUT ステートメントで出力行を解放した後、出力ポインタを出力バッファの列 1 に移動します。

このプログラムを実行すると、次の行が SAS ログに出力されます。

```

----+----1----+----2
David    180
Amelia   145
Monica
Alan     210

```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

### 例 6: 現在の入力レコードをログに書き込む

ID の値が 1000 未満の場合、PUT \_INFILE\_ を実行し、現在の入力レコードを SAS ログに書き込みます。DELETE ステートメントを実行することによって、DATA ステップで TEAM データセットにオブザベーションが書き込まれないようにします。

```

data team;
input id team $ score1 score2;
if id le 1000 then
do;
put _infile_;
delete;
end;
datalines;
032 red 180 165
049 yellow 145 124
219 red 210 192
;

```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```

----+----1----+----2
219 red 210 192

```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

### 例 7: 変数が後ろに続く文字定数の書き込み時によく発生するエラーを回避する

この例では、SAS 名前リテラルとして誤って解釈されないように、PUT ステートメントを使用して文字リテラルと変数の値を書き込む方法を示します。SAS 名前リテラルとは、引用符で囲んだ文字列として表される名前トークンで、この文字列の後ろに文字 n が追加されます。

次のプログラムでは、PUT ステートメントは変数 NVAR1 の値が後ろに続く定数 'n' を書き込んでから、別の定数 'n' を書き込みます。

```
data _null_;
  n=5;
  nvar1=1;
  var1=7;
  put @1 'n' nvar1 'n';
run;
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
----+----1----+----2
n1 n
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

前述の PUT ステートメントから定数と変数の間にある空白をすべて削除すると、'n' は定数として読み込まれずに、名前リテラルと解釈されます。次の変数は、NVAR1 ではなく VAR1 として読み込まれます。最後の定数 'n' は正しく解釈されます。

```
put @1 'n'nvar1'n';
```

この PUT ステートメントを実行すると、次の行が SAS ログに書き込まれます。

```
----+----1----+----2
57 n
```

文字定数と変数の値の間に空白を追加した後、必要に応じてポインタコントロールを使用すると、誤った解釈を潜在的に発生させることなく、文字定数と変数の値の間に空白を挿入することなく書き込まれます。たとえば、次の PUT ステートメントではポインタコントロールを使用して正しい文字定数と変数の値を書き込みますが、空白は挿入されません。+(-1) と指定すると、PUT ステートメントでポインタが 1 列だけ戻ります。

```
put @1 'n' nvar1 +(-1) 'n';
```

この PUT ステートメントを実行すると、次の行が SAS ログに書き込まれます。

```
----+----1----+----2
n1n
```

### 例 8: 複数の列で構成される出力を作成する

この例では、#n 行ポインタコントロールと @n カラムポインタコントロールを使用して複数の列で構成される出力を作成します。

```
/*
 * Use #i and @j to position name and weight information into
 * four columns in column-major order. That is print down column 1
 * first, then print down column 2, etc.
 * This example highlights the need to specify # before @ because
 * # sets the column pointer to 1.
 */
```

```

data _null_;
  file print n=ps notitles header=hd;

  do i = 1 to 80 by 20;
    do j = 1 to ceil(num_students/4);
      set sashelp.class nobs=num_students;
      put #(j+3)@i name $8. ' ' +1 weight 5.1;
    end;
  end;
  stop;

hd:
  put @26 'Student Weight in Pounds' / @26 24*'-';
  return;
run;

```

このプログラムを実行すると、次の出力が作成されます。

```

Student Weight in Pounds -----
Alfred - 112.5 James - 83.0 Joyce - 50.5 Robert - 128.0 Alice - 84.0 Jane -
84.5 Judy - 90.0 Ronald - 133.0 Barbara - 98.0 Janet - 112.5 Louise - 77.0
Thomas - 85.0 Carol - 102.5 Jeffrey - 84.0 Mary - 112.0 William - 112.0 Henry -
102.5 John - 99.5 Philip - 150.0

```

## 関連項目:

### ステートメント:

- [“FILE ステートメント” \(75 ページ\)](#)
- [“PUT ステートメント、カラム” \(336 ページ\)](#)
- [“PUT ステートメント、フォーマット” \(338 ページ\)](#)
- [“PUT ステートメント、リスト” \(343 ページ\)](#)
- [“PUT ステートメント、名前付き” \(347 ページ\)](#)

### システムオプション:

- [“LINESIZE= System Option” \(SAS Viya System Options: Reference\)](#)
- [“PAGESIZE= System Option” \(SAS Viya System Options: Reference\)](#)

---

## PUT ステートメント、カラム

変数の値を出力行の指定した列に書き込みます。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 実行

---

## 構文

```
PUT variable start-column <- end-column>
<.decimal-places> <@ | @@>;
```

## 引数

### *variable*

変数の名前を指定します。指定した変数の値が書き込まれます。

### *start-column*

出力行での、値を書き込むフィールドの開始列を指定します。

### *-end-column*

この値のフィールドの最終列を指定します。

**ヒント** 出力行に必要な列数が 1 つの場合、*end-column* の指定を省略することができます。

**例** *end-column* が省略されているため、文字変数 GENDER の値は列 16 にのみ存在します。  

```
put name 1-10 gender 16;
```

### *.decimal-places*

数値で使用する小数点以下の桁数を指定します。

**範囲** 正の整数

**ヒント** *d* の値に 0 を指定したり、*d* の値を省略する場合は、小数点なしの値が書き込まれます。

**例** [“例: PUT ステートメントでカラム出力の使用” \(337 ページ\)](#)

### @ | @@

次の PUT ステートメントの実行時に使用できるように出力行を保持します。このラインホールド指定子は、後置@および後置@@と呼ばれます。

**要件** 後置@または後置@@は、PUT ステートメントの最後の項目として指定する必要があります。

**参照項目:** [“ラインホールド指定子の使用” \(327 ページ\)](#)

## 詳細

カラム出力では、列番号は出力行内で各変数値が占める位置を示します。指定した列数より少ない列数で値を出力できる場合、文字変数は左寄せで、数値変数は右寄せで表示されます。

1 つの PUT ステートメントでの列の指定数に制限はありません。出力行のどの位置でも値を書き込むことができます。また、同じステートメントですでに値を書き込んだ列の上に値を上書きすることもできます。1 つの PUT ステートメントで、列の出力を他の出力スタイルと組み合わせられます。詳細については、[“1 つの PUT ステートメントで複数の出力スタイルを使用する” \(326 ページ\)](#)を参照してください。

## 例: PUT ステートメントでカラム出力の使用

PUT ステートメントでカラム出力を使用する例を次に示します。

- 次の PUT ステートメントはカラム出力を使用します。

```
data _null;
  input name $ 1-18 score1 score2 score3;
  put name 1-20 score1 23-25 score2 28-30
    score3 33-35;
  datalines;
Joseph      11 32 76
Mitchel     13 29 82
Sue Ellen   14 27 74
;
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
----+----1----+----2----+----3----+----4
Joseph      11 32 76
Mitchel     13 29 82
Sue Ellen   14 27 74
```

注 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

文字変数 NAME の値は列 1 から出力されます。また、指定したフィールド (列 1 から 20) に左寄せで表示されます。数値変数 SCORE1 から SCORE3 の値は、各フィールドに右寄せで表示されます。

- 次のステートメントでも同じ出力行が生成されますが、SCORE1 の値が最初に書き込まれ、NAME の値が最後に書き込まれます。

```
put score1 23-25 score2 28-30
  score3 33-35 name $ 1-20;
```

- 次の DATA ステップでは、カラム出力に小数点以下の桁数を指定します。

```
data _null;
  x=11;
  y=15;
  put x 10-18 .1 y 20-28 .1;
run;
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4
          11.0  15.0
```

## 関連項目:

### ステートメント:

- [“PUT ステートメント” \(318 ページ\)](#)

---

## PUT ステートメント、フォーマット

指定した出力形式で変数の値を出力行に書き込みます。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 実行

## 構文

**PUT** <*pointer-control*> *variable format*. <@ | @@>;

**PUT** <*pointer-control*> (*variable-list*) (*format-list*) <@ | @@>;

## 引数

### *pointer-control*

指定した行または列に出力ポインタを移動させます。

参照項目: “[カラムポインタコントロール](#)” (321 ページ)

“[行ポインタコントロール](#)” (322 ページ)

例 “[例 1: フォーマット値の間に文字を書き込む](#)” (341 ページ)

### *variable*

変数の名前を指定します。指定した変数の値が書き込まれます。

### (*variable-list*)

変数のリストを指定します。指定した変数の値が書き込まれます。

要件 (*format-list*)は(*variable-list*)の後ろに指定する必要があります。

参照項目: “[変数と出力形式をグループ化する方法](#)” (341 ページ)

例 “[例 1: フォーマット値の間に文字を書き込む](#)” (341 ページ)

### *format.*

変数の値を書き込むときに使用する出力形式を指定します。デフォルトの配置を無効にするには、配置の指定を出力形式に追加します。

- L 値を左揃えします。
- C 値を中央揃えします。
- R 値を右揃えします。

ヒント 出力形式の長さには、値に加えて出力形式の指定に使用するカンマ、ドル記号、小数点、その他の特殊文字を表示するのに十分なスペースを指定してください。

例 次の PUT ステートメントは、出力形式 `dollar7.2` を使用して X の値を書き込みます。  

```
put x dollar7.2;
```

X が 100 の場合、フォーマット値は 7 列を使用します。  

```
$100.00
```

例 “[例 2: フォーマット値のデフォルトの配置を無効にする](#)” (342 ページ)

### (*format-list*)

出力形式のリストを指定します。このリストは、前に指定した変数リストの値を書き込むときに使用します。PUT ステートメントでは、*format-list* に次を指定できます。

**format.**

変数の値を書き込むときに使用する出力形式を指定します。

ヒント SAS 出力形式またはユーザー定義の出力形式のどちらかを指定できます。

参照項目: [SAS Viya Formats and Informats: Reference](#)

**pointer-control**

値の読み込み位置を指示するポインタコントロール(@、#、/、+、OVERPRINT)のうちの1つを指定します。

例 “例 1: フォーマット値の間に文字を書き込む” (341 ページ)

**character-string**

フォーマット値の間に追加する1つまたは複数の文字を指定します。

例 次のステートメントは、フォーマット値 CODE1、CODE2、CODE3 の間にハイフンを追加します。

```
put bldg $ (code1 code2 code3) (3. '-');
```

例 “例 1: フォーマット値の間に文字を書き込む” (341 ページ)

**n\***

出力形式リストで n\* に続けて指定した出力形式を n 回繰り返します。

制限事項 (format-list) は (variable-list) の後ろに指定する必要があります。

参照項目: “変数と出力形式をグループ化する方法” (341 ページ)

例 次のステートメントは、7.2 出力形式を使用して GRADES1、GRADES2、GRADES3 を書き込み、5.2 出力形式を使用して GRADES4、GRADES5 を書き込みます。

```
put (grades1-grades5) (3*7.2, 2*5.2);
```

**@ | @@**

次の PUT ステートメントの実行時に使用できるように出力行を保持します。出力行は、DATA ステップの反復間でも保持されます。このラインホールド指定子は、後置@ および 後置@@ と呼ばれます。

制限事項 後置@ または 後置@@ は、PUT ステートメントの最後の項目として指定する必要があります。

参照項目: “ラインホールド指定子の使用” (327 ページ)

**詳細****フォーマット出力の使用**

フォーマット出力では、出力行を指定するために、変数名とその変数の値を書き込むときに使用する出力形式をリストします。SAS 出力形式またはユーザー定義の出力形式を使用して、変数の値の出力形式を制御することができます。SAS 出力形式の詳細については、“[Definition of Formats](#)” ([SAS Viya Formats and Informats: Reference](#)) を参照してください。



フォーマット出力では、PUT ステートメントでは、変数の値を書き込むときに使用する出力形式を変数名の後ろに指定します。値の間に空白は自動的に追加されません。指定した列数より少ない列数で値を出力できる場合、出力形式の幅で指定されたフィールド内に文字の値は左寄せ、数値は右寄せで表示されます。

フォーマット出力とポインタコントロールを組み合わせると、変数の値を書き込む行と列の位置を正確に指定できます。次の PUT ステートメントは、dollar7.2 出力形式を使用して、列 12 の位置から中央揃えで X の値を表示します。

```
put @12 x dollar7.2-c;
```

### 変数と出力形式をグループ化する方法

特定のパターンで出力行に値を書き込む場合、出力形式リストを使用するとコードを書く時間を短縮することができます。出力形式リストは、出力形式を空白またはカンマで区切り、丸かっこで囲みます。出力形式リストは、丸かっこで囲んだ変数名のリストの後に指定する必要があります。

たとえば、次のステートメントは、出力形式リストを使用して SCORE1 から SCORE5 までの 5 つの変数を順に書き込みます。各変数の値の出力に列を 4 つ使用します。また、値の間に空白は追加しません。

```
put (score1-score5) (4. 4. 4. 4. 4.);
```

前述のステートメントは次のように簡略化できます。

```
put (score1-score5) (4.);
```

出力形式リストには、任意のポインタコントロール(@、#、/、+、OVERPRINT)、n\*、および文字列を挿入できます。PUT ステートメントには必要な数の出力形式リストを使用できますが、出力形式リストはネストさせないでください。変数リストにあるすべての値が書き込まれると、PUT ステートメントは出力形式リストに残っている指示を無視します。例については、“[例 3: 必要な数以上の出力形式が指定されている場合](#)” (342 ページ)を参照してください。

配列に含まれるすべての要素への参照も指定できます。出力形式リストの前に、(array-name {\*})を指定します。ただし、この場合、\_TEMPORARY\_配列内の要素を指定することはできません。次の PUT ステートメントは、配列名と出力形式リストを指定します。

```
put (array1{*}) (4.);
```

配列の参照方法の詳細については、“[配列](#)” (329 ページ)を参照してください。

## 例

### 例 1: フォーマット値の間に文字を書き込む

次の例では、一部の値をフォーマットし、変数 BLDG と ROOM の間に-(ハイフン)を 1 つ書き込みます。

```
data _null_;
  input name & $15. bldg $ room;
  put name @20 (bldg room) ($1. "-" 3.);
  datalines;
Bill Perkins J 126
Sydney Riley C 219
;
```

次の行が SAS ログに書き込まれます。

Bill Perkins J-126  
 Sydney Riley C-219

### 例 2: フォーマット値のデフォルトの配置を無効にする

次の例では出力形式に配置方法を指定します。

```
data _null_;
  input name $ 1-12 score1 score2 score3;
  put name $12.-r +3 score1 3. score2 3.
     score3 4.;
  datalines;
Joseph      11 32 76
Mitchel     13 29 82
Sue Ellen   14 27 74
;
```

次の行が SAS ログに書き込まれます。<sup>1</sup>

```
----+----1----+----2----+----3----+----4
      Joseph 11 32 76
      Mitchel 13 29 82
      Sue Ellen 14 27 74
```

文字変数 NAME の値はフォーマットフィールドに右寄せで表示されています。(文字変数の場合、デフォルト設定は左寄せです。)

### 例 3: 必要な数以上の出力形式が指定されている場合

次の出力形式リストには、PUT ステートメントの実行時に必要となる数よりも多い出力形式が指定されています。

```
data _null_;
  input x y z;
  put (x y z) (2.,+1);
  datalines;
2 24 36
0 20 30
;
```

この PUT ステートメントでは、出力形式 2. を使用して X の値を書き込みます。カラムポインタコントロールは +1 と指定されているため、ポインタを 1 列だけ前方(右)に移動します。次に、出力形式 2. を使用して Y の値を書き込みます。カラムポインタコントロールは +1 と指定されているため、再度ポインタを 1 列だけ前方(右)に移動します。最後に、出力形式 2. を使用して Z の値を書き込みます。3 回目の繰り返しでは、PUT ステートメントはポインタコントロールの移動を指示する +1 を無視します。

次の行が SAS ログに書き込まれます。<sup>2</sup>

```
----+----1----+
2 24 36
0 20 30
```

### 関連項目:

### ステートメント:

<sup>1</sup> 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。  
<sup>2</sup> 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

- [“PUT ステートメント” \(318 ページ\)](#)

---

## PUT ステートメント、リスト

変数値と指定した文字列を出力行に書き込みます。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 実行

---

### 構文

**PUT** <*pointer-control*> *variable* <@ | @@>;

**PUT** <*pointer-control*> <*n*\*> 'character-string' <@ | @@>;

**PUT** <*pointer-control*> *variable* <: | ~> *format*.<@ | @@>;

### 引数

#### *pointer-control*

指定した行または列に出力ポインタを移動させます。

参照項目: [“コラムポインタコントロール” \(321 ページ\)](#)

[“行ポインタコントロール” \(322 ページ\)](#)

例 [“例 2: 文字列と変数値の書き込み” \(346 ページ\)](#)

#### *variable*

変数の名前を指定します。指定した変数の値が書き込まれます。

例 [“例 1: リスト出力の値の書き込み” \(346 ページ\)](#)

#### *n*\*

後ろに指定する文字列を *n* 回繰り返します。

例 次のステートメントは、1 つの行にアンダースコアを 132 回書き込みます。

```
put 132*! _;
```

#### 'character-string'

書き込む文字列を一重引用符で囲んで指定します。

**操作** 指定した文字列全体を書き込むスペースが現在の行に残されていない場合、指定した文字列を保留して、現在の行を書き込みます。次に、新しい行の列 1 からこの保留した文字列を書き込みます。詳細については、[“ポインタが行の終わりを超える場合” \(328 ページ\)](#)を参照してください。

**ヒント** 誤って解釈されるのを防ぐため、PUT ステートメントで使用する終了引用符の後ろには常に空白を追加してください。

---

引用符の後ろに X がある場合、指定した文字列が 16 進定数であると判断されます。

間に空白を含まない連続した 2 つの一重引用符(')または二重引用符(")をテキスト文字列として指定すると、一重引用符(')または二重引用符(")がそれぞれ生成されます。

参照項目: “リスト出力のスペース設定方法” (345 ページ)

例 “例 2: 文字列と変数値の書き込み” (346 ページ)

:

PUT ステートメントで変数値を書き込むときに使用する出力形式を指定できます。値の先頭または末尾にある空白は削除されます。また、値の後ろに空白が 1 つ追加されます。

要件 出力形式を指定する必要があります。

参照項目: “修飾リスト出力とフォーマット出力の違い” (345 ページ)

例 “例 3: 修飾リスト出力(:)を用いた値の書き込み” (347 ページ)

~

PUT ステートメントで変数値を書き込むときに使用する出力形式を指定できます。フォーマット値に区切り文字が含まれていない場合でも、フォーマット値を引用符で囲んで表示します。値の先頭または末尾にある空白は削除されます。また、値の後ろに空白が 1 つ追加されます。デフォルトでは、文字変数の欠損値は空白(" ")として書き込まれ、数値の欠損値はピリオド(" ")として書き込まれます。

要件 FILE ステートメントに DSD オプションを指定する必要があります。

例 “例 4: 修飾リスト出力(~)を用いた値の書き込み” (347 ページ)

### **format.**

データの値を書き込むときに使用する出力形式を指定します。

ヒント SAS 出力形式またはユーザー定義の出力形式のどちらかを指定できます。次を参照してください。SAS *Viya Formats and Informats: Reference*

例 “例 3: 修飾リスト出力(:)を用いた値の書き込み” (347 ページ)

### **@ | @@**

次の PUT ステートメントの実行時に使用できるように出力行を保持します。このラインホールド指定子は、後置@および後置@@と呼ばれます。

制限事項 後置@または後置@@は、PUT ステートメントの最後の項目として指定する必要があります。

参照項目: “ラインホールド指定子の使用” (327 ページ)

## 詳細

### リスト出力の使用

リスト出力を使用する場合、値を書き込む変数名をリストするか、文字列を引用符で囲んで指定します。PUT ステートメントは変数値を書き込んだ後、空白を 1 つ挿入してから次の値を書き込みます。数値の欠損値は 1 つのピリオドとして書き込まれます。文字列はフィールドに左寄せで表示されます。また文字列の先頭と末尾にある空白は削除されます。それぞれの値の後ろに追加される空白以外に、空白を挿入するには、リスト出力ではなく、フォーマット出力またはカラム出力を使用します。

リスト入力には、次の 2 種類があります。

- 単純リスト出力
- 修飾リスト出力

修飾リスト出力では、出力形式を指定して変数値を書き込む方法を制御できるため、PUT ステートメントの用途が広がります。“例 3: 修飾リスト出力(:)を用いた値の書き込み” (347 ページ)を参照してください。

### リスト出力のスペース設定方法

リスト出力は、変数値や文字列を書き込む場合に、各種のスペース設定を使用します。リスト出力を使用して変数値を書き込む場合、空白が自動的に挿入されます。そのため、出力ポインタは変数値の後の 2 列目に移動します。ただし、文字列を書き込む場合は空白は自動的に挿入されません。この場合、出力ポインタは文字列の最終文字のすぐ後の列に移動します。

文字列や変数値の書き込み時にスペース設定の問題が発生しないように、文字列の最終文字として空白を 1 つ追加することをお勧めします。変数値の後ろに句読点を提供する文字列を続ける場合、出力ポインタを戻す必要があります。出力ポインタを戻すことで、出力行に不要な空白が挿入されることを防げます。“例 2: 文字列と変数値の書き込み” (346 ページ)を参照してください。

### 修飾リスト出力とフォーマット出力の違い

リスト出力とフォーマット出力は、異なる方式を使用して、変数値の書き込み後にポインタをどれだけ移動するかを決定します。そのため、出力形式を使用する修飾リスト出力とフォーマット出力で、出力行の結果が異なります。修飾リスト出力では、値を書き込み空白を 1 つ挿入してから、ポインタを次の列に移動します。フォーマット出力では、値が指定した出力形式の長さに満たない場合でも、出力形式の長さの分だけポインタを移動します。ポインタは、空白を挿入せずに、次の列に移動します。

次の DATA ステップは、修飾リスト出力を使用して各出力行を書き込みます。

```
data _null_;
  input x y;
  put x : comma10.2 y : 7.2;
  datalines;
2353.20 7.10
6231 121
;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2
2,353.20 7.10
6,231.00 121.00
```

比較するため、次の例ではフォーマット出力を使用します。

```
put x comma10.2 y 7.2;
```

次の行が SAS ログに書き込まれます。列内の値の位置が揃えられています。

```
----+----1----+----2
      2,353.20  7.10
      6,231.00 121.00
```

## 例

### 例 1: リスト出力の値の書き込み

次の DATA ステップは、リスト出力を伴う PUT ステートメントを使用して、変数値を SAS ログに書き込みます。

```
data _null_;
  input name $ 1-10 sex $ 12 age 15-16;
  put name sex age;
  datalines;
Joseph M 13
Mitchel M 14
Sue Ellen F 11
;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4
Joseph M 13
Mitchel M 14
Sue Ellen F 11
```

デフォルトでは、文字変数 NAME の値はフィールド内に左寄せで表示されます。

### 例 2: 文字列と変数値の書き込み

次の PUT ステートメントは、文字列の末尾に 1 つの空白を追加し、出力ポインタを逆報告に移動することにより、変数 STARTWGHT の後の出力行内に不要な空白が表示されないようにします。

```
data _null_;
  input idno name $ startwght;
  put name 'weighs ' startwght +(-1) '!';
  datalines;
032 David 180
049 Amelia 145
219 Alan 210
;
```

次の行が SAS ログに書き込まれます。

```
David weighs 180.
Amelia weighs 145.
Alan weighs 210.
```

文字列の最後にブランクが追加されるため、ポインタの位置が変わります。このブランクによって、文字列と後続の変数値が区切られます。ポインタコントロールを+(-1)と指定しているため、ポインタが後方(左)に移動し、STARTWGHT の値とピリオドの間の不要なブランクが削除されます。

**例 3: 修飾リスト出力(:)を用いた値の書き込み**

次の DATA ステップは、:引数を用いた修飾リスト出力を使用して、複数の変数値を 1 つの出力行に書き込みます。

```
data _null_;
  input salesrep : $10. tot : comma6. date : date9.;
  put 'Week of ' date : worddate15.
      salesrep : $12. 'sales were '
      tot : dollar9. + (-1) '!';
  datalines;
Wong 15,300 12OCT2004
Hoffman 9,600 12OCT2004
;
```

次の行が SAS ログに書き込まれます。

```
Week of Oct 12, 2004 Wong sales were $15,300.
Week of Oct 12, 2004 Hoffman sales were $9,600.
```

**例 4: 修飾リスト出力(~)を用いた値の書き込み**

次の DATA ステップは、~引数を用いた修飾リスト出力を使用して、複数の変数値を 1 つの出力行に書き込みます。

```
data _null_;
  input salesrep : $10. tot : comma6. date : date9.;
  file log delimiter=" " dsd;
  put 'Week of ' date ~ worddate15.
      salesrep ~ $12. 'sales were '
      tot ~ dollar9. + (-1) '!';
  datalines;
Wong 15,300 12OCT2004
Hoffman 9,600 12OCT2004
;
```

次の行が SAS ログに書き込まれます。

```
Week of "Oct 12, 2004" "Wong" sales were "$15,300".
Week of "Oct 12, 2004" "Hoffman" sales were "$9,600".
```

**関連項目:****ステートメント:**

- [“PUT ステートメント” \(318 ページ\)](#)
- [“PUT ステートメント、フォーマット” \(338 ページ\)](#)

---

**PUT ステートメント、名前付き**

変数名と等号の後に変数値を書き込みます。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 実行

---

## 構文

**PUT** <pointer-control> variable= <format.> <@ | @@>;

**PUT** variable= start-column <-end-column> <.decimal-places> <@ | @@>;

## 引数

### **pointer-control**

出力バッファ内の指定した行または列に出力ポインタを移動させます。

参照項目: “[カラムポインタコントロール](#)” (321 ページ)

“[行ポインタコントロール](#)” (322 ページ)

### **variable=**

変数を次の形式で指定します。指定した変数値が PUT ステートメントで書き込まれます。

variable=value

### **format.**

変数の値を書き込むときに使用する出力形式を指定します。

**ヒント** 出力形式の長さには、値に加えて出力形式の指定に使用するカンマ、ドル記号、小数点、その他の特殊文字を表示するのに十分なスペースを指定してください。

参照項目: “[名前付き出力のフォーマット](#)” (349 ページ)

**例** 次の PUT ステートメントは、出力形式 DOLLAR7.2 を使用して X の値を書き込みます。  
put x= dollar7.2;

X=100 の場合、フォーマット値は 7 列使用します。  
X=\$100.00

### **start-column**

変数名、等号、値を書き込む出力行のフィールドの開始列位置を指定します。

### **- end-column**

値のフィールドの最終列位置を指定します。

**ヒント** 変数名、等号、値の表示に必要な列数が指定した列数より多い場合、値を切り捨てずに列の最終位置を越えて書き込みます。そのため、次の値の開始位置との間に十分なスペースを確保する必要があります。

### **.decimal-places**

数値で使用する小数点以下の桁数を指定します。d に 0 を指定したり、d の値を省略する場合は、小数点なしの値が書き込まれます。

範囲 正の整数

### **@ | @@**

次の PUT ステートメントの実行時に使用できるように出力行を保持します。出力行は、DATA ステップの反復間でも保持されます。このラインホールド指定子は、[後置@](#)および[後置@@](#)と呼ばれます。



**制限事項** 後置@または後置@@は、PUT ステートメントの最後の項目として指定する必要があります。

**参照項目:** “ラインホールド指定子の使用” (327 ページ)

## 詳細

### 名前付き出力の使用

名前付き出力を使用するには、PUT ステートメントに変数名の後に等号を指定します。変数名と値の表示位置を指定するには、リスト出力、カラム出力、フォーマット出力のいずれかを使用できます。各変数の間に自動的に空白を挿入するには、リスト出力を使用します。列の出力をそろえるには、ポインタコントロールと列を指定します。

### 名前付き出力のフォーマット

SAS 出力形式またはユーザー定義の出力形式のいずれかを指定して、変数値の出力を制御することができます。出力形式の幅には、変数名と等号の表示に必要な列は含まれません。フォーマット値を揃えるため、変数値は先頭にある空白は削除され、等号のすぐ後ろに書き込まれます。名前付きでないフォーマット出力のように、指定した幅の右側に揃えることはありません。

SAS 出力形式の詳細については、“[Definition of Formats](#)” (*SAS Viya Formats and Informats: Reference*)を参照してください。

## 例: PUT ステートメントで名前付き出力を使用する

PUT ステートメントで名前付き出力を使用する例を次に示します。

- この PUT ステートメントでは、名前付き出力とカラムポインタコントロールを組み合わせて出力を配置します。

```
data _null_;
  input name $ 1-18 score1 score2 score3;
  put name = @20 score1= score3= ;
  datalines;
Joseph      11 32 76
Mitchel     13 29 82
Sue Ellen   14 27 74
;
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
----+----1----+----2----+----3----+----4
NAME=Joseph   SCORE1=11 SCORE3=76
NAME=Mitchel  SCORE1=13 SCORE3=82
NAME=Sue Ellen SCORE1=14 SCORE3=74
```

- この例では、変数 AMOUNT の出力形式を指定します。

```
put item= @25 amount= dollar12.2;
```

ITEM の値が binders で AMOUNT の値が 153.25 の場合、出力行は次のように表示されます。

```
----+----1----+----2----+----3----+----4
ITEM=binders   AMOUNT=$153.25
```

**関連項目:****ステートメント:**

- [“PUT ステートメント”\(318 ページ\)](#)

---

**PUTLOG ステートメント**

SAS ログにメッセージを書き込みます。

**該当要素:** DATA ステップ

**カテゴリ:** アクション  
CAS

**種類:** 実行

---

**構文**

**PUTLOG** *message*;

**引数*****message***

SAS ログに書き込むメッセージを指定します。*message* には、文字リテラル (一重引用符で囲む)、変数名、出力形式、ポインタコントロールを指定できません。

**ヒント** メッセージテキストの前に、WARNING、MESSAGE、NOTE などを追加すると、ログの出力内容を判別しやすくなります。

---

**詳細**

PUTLOG ステートメントでは、指定したメッセージを SAS ログに書き込みます。現在のファイル出力先に影響せずに SAS ログに出力するため、PUTLOG ステートメントはマクロを生成するコードの使用時にも便利です。

**比較**

PUTLOG ステートメントは ERROR ステートメントに似ていますが、PUTLOG ステートメントを実行しても自動変数 `_ERROR_` の値は 1 に設定されません。

**例: PUTLOG ステートメントを使用してメッセージを SAS ログに書き込む**

次のプログラムでは、`computeAverage92` マクロを作成します。このマクロは、平均点の算出、入力データの検証、PUTLOG ステートメントを使用した SAS ログへのエラーメッセージの書き込みを行います。DATA ステップでは、PUTLOG ステートメントを使用して警告メッセージをログに書き込みます。

```
data ExamScores;
  input Name $ 1-16 Score1 Score2 Score3;
  datalines;
Sullivan,James 86 92 88
```

```

Martinez, Maria 95 91 92
Guzik, Eugene 99 98 .
Schultz, John 90 87 93
van Dyke, Sylvia 98 . 91
Tan, Carol 93 85 85
;

filename outfile 'path-to-your-output-file';
/* Create a macro that computes the average score, validates */
/* input data, and uses PUTLOG to write error messages to the */
/* SAS log. */
%macro computeAverage92(s1, s2, s3, avg);
  if &s1 < 0 or &s2 < 0 or &s3 < 0 then
  do;
    putlog 'ERROR: Invalid score data ' &s1= &s2= &s3=;
    &avg = .;
  end;
  else
    &avg = mean(&s1, &s2, &s3);
%mend;
data _null_;
set ExamScores;
file outfile;
%computeAverage92(Score1, Score2, Score3, AverageScore);
put name Score1 Score2 Score3 AverageScore;
/* Use PUTLOG to write a warning message to the SAS log. */
if AverageScore < 92 then
  putlog 'WARNING: Score below the minimum ' name= AverageScore= 5.2;
run;

proc print;
run;

```

次の行が SAS ログに書き出されます。

```

WARNING: Score below the minimum Name=Sullivan, James AverageScore=88.67 ERROR: Invalid score
data Score1=99 Score2=98 Score3=.WARNING: Score below the minimum Name=Guzik, Eugene
AverageScore=.WARNING: Score below the minimum Name=Schultz, John AverageScore=90.00 ERROR:
Invalid score data Score1=98 Score2=.Score3=91 WARNING: Score below the minimum Name=van
Dyke, Sylvia AverageScore=.WARNING: Score below the minimum Name=Tan, Carol AverageScore=87.67

```

次の出力ファイルが作成されます。

## アウトプット 2.27 個人別のテスト結果

Obs	Name	Score1	Score2	Score3
1	Sullivan, James	86	92	88
2	Martinez, Maria	95	91	92
3	Guzik, Eugene	99	98	.
4	Schultz, John	90	87	93
5	van Dyke, Sylvia	98	.	91
6	Tan, Carol	93	85	85

## 関連項目:

## ステートメント:

- [“ERROR ステートメント” \(73 ページ\)](#)

---

**REDIRECT ステートメント**

ストアドプログラムを実行するときに、別の入力 SAS データセットまたは出力 SAS データセットを参照します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。

**要件** DATA ステートメントに PGM=オプションを指定する必要があります。

**構文**

**REDIRECT INPUT | OUTPUT** *old-name-1=new-name-1 <...old-name-n=new-name-n>*;

**引数****INPUT | OUTPUT**

入力データセットまたは出力データセットのどちらをリダイレクトするか指定します。INPUT を指定すると、プログラム内の入力データセット名が別の SAS データセット名に割り当てられます。OUTPUT を指定すると、出力データセット名が別の SAS データセット名に割り当てられます。

**old-name**

プログラム内の入力データセットまたは出力データセット名を指定します。

**new-name**

現在の実行で処理する入力データセットまたは出力データセット名を指定します。

**詳細**

REDIRECT ステートメントはストアプログラムを実行する場合にのみ使用できます。

**注意:**

入力データセットをリダイレクトするときは注意が必要です。REDIRECT ステートメントで読み込む入力データセット内の変数の数や属性は、ソースコード内の MERGE、SET、MODIFY、または UPDATE の各ステートメントの入力データセットで指定した変数の数や属性と一致している必要があります。変数の種類の属性が異なる場合、ストアプログラムの処理が中止し、該当するエラーメッセージが SAS ログに書き込まれます。変数の長さ属性が異なる場合、リダイレクトされるデータセットに含まれる変数の長さは、ソースコードのデータセットに含まれる変数の長さによって決定します。リダイレクト先のデータセットのほうが変数の数が多い場合は、ストアプログラムの処理が中止し、該当するエラーメッセージが SAS ログに書き込まれます。

**ヒント** REDIRECT ステートメントで読み込む入力データセット内の変数の数が、プログラムをコンパイルするために使用するデータセット内の変数の数よりも多い場合、ストアプログラムに DROP または KEEP データセットオプションを追加できます。

**比較**

REDIRECT ステートメントは、SAS データセットに対してのみ適用されます。外部ファイルに指定されている入力データセットまたは出力データセットにリダイレクトするには、FILENAME ステートメントを使用してソースプログラムのファイル参照名(fileref)と外部ファイルを関連付けます。

**例: ストアドプログラムの実行**

この例では、STORED.SAMPLE というストアプログラムを実行します。REDIRECT ステートメントでは、入力データソース BASE.SAMPLE を指定します。このプログラムの実行で使用する出力データセットは、SUMS.SAMPLE というデータセットにリダイレクトした後、格納されます。

```
libname stored 'SAS-library';
libname base 'SAS-library';
libname sums 'SAS-library';
data pgm=stored.sample;
  redirect input in.sample=base.sample;
  redirect output out.sample=sums.sample;
run;
```

**関連項目:****ステートメント:**

- [“DATA ステートメント” \(47 ページ\)](#)

---

## REMOVE ステートメント

SAS データセットからオブザベーションを削除します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。  
必ず MODIFY ステートメントと組み合わせて使用します。

---

### 構文

**REMOVE** <*data-set-name(s)*>;

### 引数なし

引数を指定しない場合、REMOVE ステートメントは DATA ステートメントに指定されているすべてのデータセットから現在のオブザベーションを削除します。

### 引数

#### *data-set-name*

オブザベーションを削除するデータセットを指定します。

**制限事項** このデータセット名は、DATA ステートメントおよび 1 つ以上の MODIFY ステートメントにも指定する必要があります。

**ヒント** データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

---

### 詳細

オブザベーションは、データセットを管理するエンジンによって、物理的または論理的に削除されます。REMOVE ステートメントの使用は、オブザベーションのデフォルトの置き換えより優先されます。DATA ステップで REMOVE ステートメントを使用する場合、DATA ステップのすべての出力を明示的にプログラムする必要があります。

### 比較

- OUTPUT、REPLACE、REMOVE の各ステートメントの使用は、DATA ステップの最後に実行されるデフォルトの書き込みアクションより優先されます。(OUTPUT がデフォルトのアクションです。ただし、MODIFY ステートメントを使用すると REPLACE がデフォルトのアクションになります。)DATA ステップでこれらのステートメントのいずれかを使用する場合、新しいオブザベーションのすべての出力を明示的にプログラムする必要があります。
- OUTPUT、REPLACE、REMOVE の各ステートメントは互いに独立して動作します。順序が論理的に正しければ、同じオブザベーションに複数のステートメントを適用できます。

- 特定のオブザベーションに対して OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を実行する場合、オブザベーションのポインタの位置を正しく保つため、OUTPUT ステートメントを最後に実行します。
- REMOVE ステートメントは物理的または論理的な削除が実行できるため、すべての SAS データセットエンジンにおいて、REMOVE と MODIFY ステートメントの併用が可能です。DELETE ステートメントおよびサブセット化 IF ステートメントは、どちらも物理的な削除のみを実行します。そのため、エンジンによっては、MODIFY ステートメントとこれらのステートメントを併用できない場合があります。

## 例: データセットからオブザベーションを削除する

この例では、SAS データセットからオブザベーションを 1 つ削除します。

```
libname perm 'SAS-library';

data perm.accounts;
  input AcctNumber Credit;
  datalines;
1001 1500
1002 4900
1003 3000
;
data perm.accounts;
  modify perm.accounts;
  if AcctNumber=1002 then remove;
run;
proc print data=perm.accounts;
  title 'Edited Data Set';
run;
```

PROC PRINT ステートメントの実行結果を次に示します。

アウトプット 2.28 編集したデータセット

Edited Data Set		
Obs	AcctNumber	Credit
1	1001	1500
3	1003	3000

## 関連項目:

### ステートメント:

- [“DELETE ステートメント” \(60 ページ\)](#)
- [“IF ステートメント、サブセット化” \(134 ページ\)](#)
- [“MODIFY ステートメント” \(289 ページ\)](#)
- [“OUTPUT ステートメント” \(314 ページ\)](#)

- [“REPLACE ステートメント” \(357 ページ\)](#)

---

## RENAME ステートメント

出力 SAS データセットの変数に新しい名前を指定します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
情報

**種類:** 宣言

---

### 構文

**RENAME** *old-name-1=new-name-1* <...*old-name-n=new-name-n*>;

### 引数

#### **old-name**

入力データセットに含まれる変数名または変数リスト、または現在の DATA ステップで新しく作成する変数の名前を指定します。

#### **new-name**

出力データセットで使用する変数名またはリストを指定します。

### 詳細

RENAME ステートメントでは、1 つまたは複数の変数、1 つのリスト内の変数、または変数と変数リストの組み合わせに対して名前を変更できます。新しい変数名は、出力データセットのみに書き込まれます。現在の DATA ステップのプログラムステートメントには変更前の変数名を使用してください。RENAME ステートメントはすべての出力データセットに適用されます。

注: RENAME ステートメントは、出力モードで開かれるデータセットのみに影響します。

### 比較

- PROC ステップでは RENAME ステートメントを使用することはできませんが、RENAME=データセットオプションを使用することができます。
- RENAME=データセットオプションを使用すると、名前を変更する変数を入力データセットまたは出力データセットごとに指定できます。処理を実行する前に変数の名前を変更するには、入力データセットにこのオプションを使用します。
- 出力データセットに RENAME=データセットオプションを使用する場合、現在の DATA ステップのプログラムステートメントでは変更前の変数名を使用する必要があります。出力データセットが作成されると、新しい変数名を使用できるようになります。
- SET ステートメントに RENAME=データセットオプションを指定すると、入力データセットの変数名を変更できます。現在の DATA ステップのプログラムステートメントでも新しい名前を使用できるようになります。



- ファイル管理として変数名を変更する場合は、DATASETS プロシジャを使用するか、SAS Studio から変数にアクセスします。この方法は簡単に実行できます。また、DATA ステップを処理する必要はありません。

### 例: データセット変数の名前を変更する

- これらの例では、RENAME ステートメントを使用して変数名を変更する場合の正しい構文を示します。

```
rename street=address;
rename time1=temp1 time2=temp2 time3=temp3;
rename name=Firstname score1-score3=Newscore1-Newscore3;
```

- この例では、変更する前の変数名がプログラムステートメントで使用されています。出力データセットで、変数 Olddept が Newdept に、そして、変数 Oldaccount が ewaccount に名前が変更されます。

```
rename Olddept=Newdept Oldaccount=Newaccount;
if Oldaccount>5000;
keep Olddept Oldaccount items volume;
```

- この例では、変更する前の変数名 OLDACCNT がプログラムステートメントで使用されています。ただし、DATA ステートメントでは新しい変数名 NEWACCNT が使用されています。これは、KEEP=データセットオプションが適用される前に RENAME ステートメントが適用されるからです。

```
data market(keep=newdept newaccnt items volume);
  rename olddept=newdept oldaccnt=newaccnt;
  set sales;
  if oldaccnt>5000;
run;
```

- 次の例では、変数と変数リストの両方を使用して変数の名前を変更します。新しい変数名は出力データセットに表示されます。

```
data temp;
  input (score1-score3) (2.,+1) name $;
  rename name=Firstname
         score1-score3=Newscore1-Newscore3;
  datalines;
12 24 36 Lisa
22 44 66 Fran
;
```

### 関連項目:

#### データセットオプション:

- [“RENAME= Data Set Option” \(SAS Viya Data Set Options: Reference\)](#)

---

## REPLACE ステートメント

同じ場所にあるオブザベーションを置換します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション

種類: 実行

制限事項: このステートメントは CAS Server ではサポートされていません。  
必ず MODIFY ステートメントと組み合わせて使用します。

## 構文

```
REPLACE <data-set-name-1> <...data-set-name-n>;
```

### 引数なし

引数を指定しない場合、REPLACE ステートメントは、DATA ステートメントに指定したすべてのデータセットに対して、オブザベーションの読み込み元の物理的な場所に現在のオブザベーションを書き込みます。

### 引数

#### *data-set-name*

オブザベーションを書き込むデータセットを指定します。

要件 このデータセット名は、DATA ステートメントおよび 1 つ以上の MODIFY ステートメントにも指定する必要があります。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

## 詳細

明示的な REPLACE ステートメントの使用は、オブザベーションのデフォルトの置き換えより優先されます。DATA ステップで REPLACE ステートメントを使用する場合、DATA ステップのすべての出力をプログラムする必要があります。

## 比較

- OUTPUT、REPLACE、REMOVE の各ステートメントの使用は、DATA ステップの最後に実行されるデフォルトの書き込みアクションより優先されます。(OUTPUT がデフォルトのアクションです。ただし、MODIFY ステートメントを使用すると REPLACE がデフォルトのアクションになります。)DATA ステップでこれらのステートメントのいずれかを使用する場合、このステップの新しいオブザベーションの出力を明示的にプログラムする必要があります。
- OUTPUT、REPLACE、REMOVE の各ステートメントは互いに独立して動作します。順序が論理的に正しければ、同じオブザベーションに複数のステートメントを適用できます。
- 特定のオブザベーションに対して OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を実行する場合、オブザベーションのポインタの位置を正しく保つため、OUTPUT ステートメントを最後に実行します。
- REPLACE はオブザベーションを元の物理的な場所に書き込みます。OUTPUT は新しいオブザベーションをデータセットの最後に書き込みます。
- REPLACE ステートメントは、MODIFY ステートメントを使用した DATA ステップにのみ使用できます。OUTPUT ステートメントは、MODIFY ステートメントの有無にかかわらず指定できます。

## 例: オブザベーションの置き換え

この例では、データセット MASTER にある電話番号をデータセット TRANS にある値で置き換えます。また、データセット MASTER の最後に新しいオブザベーションが 1 つ追加されます。SYSRC 自動呼び出しマクロは、MASTER からデータを取得するたびに \_IORC\_ の値を確認します。(SYSRC は、SAS 自動呼び出しマクロライブラリのメンバです。)コードの後に、生成される SAS データセットを示します。

```
data master;
  input FirstName $ id $ PhoneNumber;
  datalines;
Kevin ABCjkh 904
Sandi defsns 905
Terry ghitDP 951
Jason jklJWM 962
;
data trans;
  input FirstName $ id $ PhoneNumber;
  datalines;
. ABCjkh 2904
. defsns 2905
Madeline mnombt 2983
;
data master;
  modify master trans;
  by id;
  /* obs found in master */
  /* change info, replace */
  if _iorc_ = %sysrc(_sok) then replace;
  /* obs not in master */
  else if _iorc_ = %sysrc(_dsenmr) then
  do;
    /* reset _error_ */
    _error_=0;
    /* reset _iorc_ */
    _iorc_=0;
    /* output obs to master */
  output;
  end;
run;
proc print data=master;
  title 'MASTER with New Phone Numbers';
run;
```

アウトプット 2.29 オブザベーションが置き換えられたデータセット

Obs	FirstName	id	PhoneNumber
1	Kevin	ABCjkh	2904
2	Sandi	defsns	2905
3	Terry	ghitDP	951
4	Jason	jklJWM	962
5	Madeline	mnombt	2983

## 関連項目:

### ステートメント:

- [“MODIFY ステートメント” \(289 ページ\)](#)
- [“OUTPUT ステートメント” \(314 ページ\)](#)
- [“REMOVE ステートメント” \(354 ページ\)](#)

---

## RESETLINE ステートメント

SAS ログ内のプログラムの行番号を 1 にリセットします。

**該当要素:** 任意の場所

**カテゴリ:** ログ制御

**種類:** 実行

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

## 構文

**RESETLINE;**

### 引数なし

RESETLINE ステートメントは、SAS ログ内のプログラムの行番号を 1 にリセットします。

## 詳細

SAS ログでは、プログラムステートメントには行番号が付きます。行番号は 1 から始まり、SAS セッションまたはバッチプログラムが終了するまで連番で採番されます。

プログラムに RESETLINE ステートメントを使用すると、プログラム行番号を再び 1 から採番します。

注: SPOOL システムオプションの使用時、最後の RESETLINE ステートメント以降にサブミットされたコード行を再度サブミットするには、%INCLUDE ステートメントのみ使用できます。

## 例: SAS ログの行番号をリセットする

次の例では、DATA ステップの間でプログラム行番号をリセットします。

```
data a;
  a=1;
run;
resetline;
data b;
  b=2;
run;
```

次の行が SAS ログに書き込まれます。

```
1 data a; 2 a=1; 3 run; NOTE: The data set WORK.A has 1 observations and 1 variables. NOTE:
DATA statement used (Total process time): real time 4.79 seconds cpu time 0.28 seconds 4
5 resetline; 1 2 data b; 3 b=2; 4 run; NOTE: The data set WORK.B has 1 observations and 1
variables. NOTE: DATA statement used (Total process time): real time 0.00 seconds cpu time
0.00 seconds
```

---

## RETAIN ステートメント

DATA ステップを繰り返す際に、INPUT ステートメントまたは割り当てステートメントで作成される変数の値を保持します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
情報

**種類:** 宣言

---

### 構文

```
RETAIN <element-list(s) <initial-value(s) | (initial-value-1) | (initial-value-list-1)>
<... element-list-n <initial-value-n | (initial-value-n) | (initial-value-list-n)> >>;
```

### 引数なし

引数を指定しない場合、DATA ステップを繰り返す際に、RETAIN ステートメントは INPUT ステートメントまたは割り当てステートメントで作成されるすべての変数の値を保持します。

### 引数

#### *element-list*

値を保持する変数名、変数リスト、配列名を指定します。

ヒ ALL、CHAR、NUMERIC を指定する場合、RETAIN ステートメント  
ン の前に定義した変数のみに影響します。  
ト

---

変数名が RETAIN ステートメントにのみ指定されている場合に、その初期値を指定しないと、この変数はデータセットに書き込まれず、変数が初期化されていないことを示す注釈が SAS ログに書き込まれます。初期値を設定すると、この変数はデータセットに書き込まれます。

### **initial-value**

直前に指定した 1 つまたは複数の要素に対し、初期値(数値または文字)を指定します。

**ヒント** *initial-value* を省略すると、初期値は欠損値になります。*Initial-value* は、その前にあるリストのすべての要素に割り当てられます。このため、変数リストのすべてのメンバに同じ初期値が割り当てられます。

**参照項目:** (*initial-value*)および(*initial-value-list*)

### **(initial-value)**

直前に指定した 1 つの要素、または要素リストにある最初の要素に対し、初期値(数値または文字)を指定します。

### **(initial-value-list)**

直前に指定したリストにある各要素に対し、初期値(数値または文字)を指定します。このリストの最初の値を要素リストの最初の変数に、2 番目の値を要素リストの 2 番目の変数に割り当てます。

要素の文字値は引用符で囲みます。1 つまたは複数の初期値を直接指定するには、次の形式を使用します。

*(initial-value(s))*

繰り返しの回数とネスト形式のサブリストを初期値に指定するには、次の形式を使用します。

<*constant-iter-value\**> <( >*constant value* | *constant-sublist*<)>

**制限事項** *initial-value-list* と *element-list* の両方を指定する場合、RETAIN ステートメントでは *element-list* を *initial-value-list* より前にリストする必要があります。

**ヒント** ブランクまたはカンマで複数の初期値を区切ることができます。

連続する整数値の範囲を指定する場合は、簡略化した表記を使用できます。増分は常に+1 になります。

変数と一時データ要素の両方に初期値を割り当てることができます。

指定した初期値の数よりも変数の数が多い場合、初期値が割り当てられなかった変数には欠損値が割り当てられ、警告メッセージが発行されます。

## **詳細**

### **DATA ステップのデフォルトの動作**

RETAIN ステートメントを使用しない場合、DATA ステップを繰り返す前に、INPUT ステートメントまたは割り当てステートメントを使用して割り当てられる変数は欠損値に設定されます。

## 初期値の割り当て

個別の変数、変数リスト、配列のメンバに初期値を指定するには、RETAIN ステートメントを使用します。RETAIN ステートメントに値が表示される場合、リスト内でこの値より前に表示される変数の初期値は表示された値に設定されます。(RETAIN ステートメントで同じ変数に異なる初期値を複数回指定した場合、最後の値が初期値として使用されます。)RETAIN ステートメントを使用すると、合計ステートメントで割り当てられた変数の値にデフォルト値 0 以外の初期値を設定することもできます。

## 指定の重複

RETAIN ステートメントで次の項目を指定すると、指定が重複することになります。次に示す項目の値は自動的に保持され、DATA ステップの繰り返しの際で使用されます。

- SET、MERGE、MODIFY、UPDATE の各ステートメントで読み込まれる変数
- 合計ステートメントで値が割り当てられた変数
- 自動変数\_N\_、\_ERROR\_、\_I\_、\_CMD\_、および\_MSG\_
- SET、MERGE、MODIFY、UPDATE の各ステートメントで END=または IN=オプションを指定するか、FILE および INFILE の各ステートメントで変数を作成するオプションを指定して作成された変数
- 一時配列で指定されたデータ要素
- ARRAY ステートメントで初期化された配列要素
- ARRAY ステートメントで任意の要素またはすべての要素に初期値が割り当てられた配列要素

ただし、RETAIN ステートメントを使用して前述の項目に初期値を割り当てることができません。自動変数\_N\_と\_ERROR\_には、初期値を設定できません。

## 比較

RETAIN ステートメントでは、DATA ステップの各繰り返しの初めに、その値が欠損値に設定されない変数を指定します。KEEP ステートメントでは、作成されるデータセットに書き込む変数を指定します。

## 例

### 例 1: 基本的な使用

- 次の RETAIN ステートメントでは、変数 MONTH1 から MONTH5 の値を保持し、DATA ステップの繰り返して使用します。

```
retain month1-month5;
```

- 次の RETAIN ステートメントでは、9 個の変数の値を保持し、その初期値を設定します。

```
retain month1-month5 1 year 0 a b c 'XYZ';
```

変数 MONTH1 から MONTH5 の初期値は 1、YEAR の初期値は 0、変数 A、B、C の初期値は文字列値 XYZ にそれぞれ設定されます。

- 次の RETAIN ステートメントは、変数 MONTH1 にのみ初期値 1 を割り当てます。

```
retain month1-month5 (1);
```

変数 MONTH2 から MONTH5 の初期値は欠損値に設定されます。

- 次の RETAIN ステートメントでは、DATA ステップの上部で定義されたすべての変数の値が保持されますが、その後定義された値は保持されません。

```
retain _all_;
```

- 次の場合、どのステートメントを実行しても変数 VAR1 から VAR4 に初期値 1 から 4 が割り当てられます。
  - retain var1-var4 (1 2 3 4);
  - retain var1-var4 (1,2,3,4);
  - retain var1-var4(1:4);

## 例 2: RETAIN ステートメントの操作概要

この例は、RETAIN ステートメントの要素として変数名と配列名を使用する方法を示しています。割り当てる初期値はかっこで囲まれているものと、かっこで囲まれていないものがあります。

```
data _null_;
  array City{3} $ City1-City3;
  array cp{3} Citypop1-Citypop3;
  retain Year Taxyear 1999 City ' '
         cp (10000,50000,100000);
  file file-specification print;
  put 'Values at beginning of DATA step:'
     / @3 _all_ /;
  input Gain;
  do i=1 to 3;
    cp{i}=cp{i}+Gain;
  end;
  put 'Values after adding Gain to city populations:'
     / @3 _all_ /;
  datalines;
5000
10000
;
```

RETAIN ステートメントで割り当てられた初期値は次のようになります。

- Year と Taxyear の初期値は 1999 になります。
- City1、City2、City3 には欠損値が割り当てられます。
- Citypop1 には 10000 が割り当てられます。
- Citypop2 には 50000 が割り当てられます。
- Citypop3 には 100000 が割り当てられます。

PUT ステートメントで書き込まれた行を次に示します。

```
Values at beginning of DATA step: City1= City2= City3= Citypop1=10000 Citypop2=50000
Citypop3=100000 Year=1999 Taxyear=1999 Gain=. i=_ERROR_=0 _N_=1 Values after adding GAIN to city
populations: City1= City2= City3= Citypop1=15000 Citypop2=55000 Citypop3=105000 Year=1999
Taxyear=1999 Gain=5000 i=4 _ERROR_=0 _N_=1 Values at beginning of DATA step: City1= City2= City3=
Citypop1=15000 Citypop2=55000 Citypop3=105000 Year=1999 Taxyear=1999 Gain=. i=_ERROR_=0
_N_=2 Values after adding GAIN to city populations: City1= City2= City3= Citypop1=25000
Citypop2=65000 Citypop3=115000 Year=1999 Taxyear=1999 Gain=10000 i=4 _ERROR_=0 _N_=2 Values at
beginning of DATA step: City1= City2= City3= Citypop1=25000 Citypop2=65000 Citypop3=115000
Year=1999 Taxyear=1999 Gain=. i=_ERROR_=0 _N_=3
```



最初の PUT ステートメントは 3 回実行されていますが、2 番目の PUT ステートメントは 2 回しか実行されていません。3 回目の INPUT ステートメントを実行して END ステートメントに到達すると、DATA ステップは終了します。

### 例 3: 複数のオブザベーションから 1 つの値を選択する

この例のデータセット ALLSCORESID には、個々の ID 番号および変数 ID に対して複数のオブザベーションが含まれています。ある 1 つの ID 値を持つ複数のオブザベーションでは、変数 GRADE の値が異なる場合があります。ここでは、新しいデータセット CLASS.BESTSCORES を作成します。作成される新しいデータセットには、ID の値ごとに 1 つのオブザベーションが含まれます。BESTSCORES データセットのオブザベーションは、ID が同じすべてのオブザベーションのうちで変数 GRADE の値が最も大きいオブザベーションになります。

```
libname class 'SAS-library';
proc sort data=class.allscores;
  by id;
run;
data class.bestscores;
  drop grade;
  set class.allscores;
  by id;
  /* Prevents HIGHEST from being reset*/
  /* to missing for each iteration. */
  retain highest;
  /* Sets HIGHEST to missing for each */
  /* different ID value. */
  if first.id then highest=.;
  /* Compares HIGHEST to GRADE in */
  /* current iteration and resets */
  /* value if GRADE is higher. */
  highest=max(highest,grade);
  if last.id then output;
run;
```

### 関連項目:

#### ステートメント:

- [“割り当てステートメント” \(28 ページ\)](#)
- [“BY ステートメント” \(33 ページ\)](#)
- [“INPUT ステートメント” \(179 ページ\)](#)

---

## RETURN ステートメント

DATA ステップ内の現在の位置でステートメントの実行を中止し、ステップで事前に定義されている位置に戻ります。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 実行

---

## 構文

**RETURN;**

### 引数なし

RETURN ステートメントは、DATA ステップ内の現在の位置でステートメントの実行を中止し、前の DATA ステップステートメントに制御を戻します。

### 詳細

戻る位置は、DATA ステップでのステートメントの実行順序によって異なります。

RETURN ステートメントは、次のステートメントと組み合わせて使用します。

- GO TO ステートメント
- FILE ステートメントの HEADER=オプション
- LINK ステートメント

DATA ステップに明示的な OUTPUT ステートメントを使用していない場合や、REMOVE ステートメントや REPLACE ステートメントが MODIFY ステートメントと組み合わせて使用されていない場合、RETURN ステートメントを使用して DATA ステップの先頭に戻ると、暗示的な OUTPUT ステートメントによって現在のオブザベーションが新しいデータセットに書き込まれます。すべての DATA ステップには、最後の実行ステートメントとして暗示的な RETURN ステートメントが含まれています。

### 例: 基本的な使用

この例では、XとYの値が等しい場合、RETURN ステートメントを実行してオブザベーションをデータセットに追加します。XとYの値が等しくない場合は、残りのステートメントを実行してからオブザベーションをデータセットに追加します。

```
data survey;
  input x y;
  if x=y then return;
  put x=y;
  datalines;
21 25
20 20
7 17
;
```

### 関連項目:

#### ステートメント:

- [“FILE ステートメント” \(75 ページ\)](#)
- [“GO TO ステートメント” \(132 ページ\)](#)
- [“LINK ステートメント” \(266 ページ\)](#)

---

## RUN ステートメント

すでに入力されている SAS ステートメントを実行します。

**該当要素:** 任意の場所

**カテゴリ:** CAS  
プログラム制御

---

### 構文

**RUN** <CANCEL>;

### 引数なし

引数を指定しない場合、RUN ステートメントはすでに入力されている SAS ステートメントを実行します。

### 引数

#### CANCEL

現在のステップを実行せずに終了します。また、ステップが実行されなかったことを示すメッセージが出力されます。

**注意:**

**CANCEL** オプションは、**DATALINES** または **DATALINES4** ステートメントを含んでいる **DATA** ステップの実行を妨げません。

**注意:**

**PROC DATASETS** で **KILL** オプションを使用する場合、**CANCEL** オプションには効果がありません。

### 詳細

SAS プログラムのステップの間に RUN ステートメントは必ずしも必要ではありませんが、RUN ステートメントを使用するとステップ境界が作成されるので、SAS ログが読みやすくなります。

### 例

#### 例 1: SAS ステートメントの実行

次の RUN ステートメントは、ステップ境界を示し、PROC PRINT ステップを実行します。

```
proc print data=report;  
  title 'Status Report';  
run;
```

#### 例 2: CANCEL オプションの使用

この例が示すように、ラインプロンプトモードのセッションに CANCEL オプションを使用すると便利です。DATA ステップの 4 行目のステートメントでは、PI に無効な値(3.14 ではなく 4.13)が指定されています。RUN ステートメントに

CANCEL オプションを指定すると、DATA ステップは終了され、その実行が回避されます。

```
data circle;
  infile file-specification;
  input radius;
  c=2*4.13*radius;
run cancel;
```

次のメッセージが SAS ログに書き込まれます。

```
WARNING: DATA step not executed at user's request.
```

---

## %RUN ステートメント

%INCLUDE \*ステートメントの後にあるソースステートメントを終了します。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

### 構文

**%RUN;**

### 引数なし

%RUN ステートメントを実行すると、(%RUN ステートメントと同じ行にある後の SAS ステートメントを含む)キーボードからの入力の読み込みを中止して、前の入力ソースの読み込みを再開します。

### 詳細

アスタリスク付きの%INCLUDE ステートメントは、キーボードからソース行を入力することを指定します。

### 比較

RUN ステートメントは、それまでに入力された DATA ステップや PROC ステップを実行します。%INCLUDE ステートメントを使用してキーボードからのデータ入力を許可している場合、%RUN ステートメントはソースステートメントのプロンプトを終了し、プログラムコントロールを元のソースプログラムに戻します。

プロンプトの形式は SAS セッションの実行方法によって異なります。キーボードからの入力は、対話型ラインモードや非対話型モードにおいて最も便利な機能ですが、バッチモードでも使用できます。バッチモードで SAS を実行している場合、ファイル参照名 SASTERM で参照されている外部ファイルに%RUN ステートメントを指定します。

### 例: キーボードからソース行を入力する

%INCLUDE ステートメントにキーボード入力ソースを要求するには、このステートメント後ろにアスタリスクを追加します。

```
%include *;
```

このステートメントを実行すると、キーボードからソース行を入力するようにプロンプトが表示されます。キーボードからコードの入力が終了したら、次のステートメントを入力して%INCLUDE を含むプログラムに処理を戻します。

```
%run;
```

## 関連項目:

### ステートメント:

- ["%INCLUDE ステートメント" \(138 ページ\)](#)
- ["RUN ステートメント" \(367 ページ\)](#)

---

## SASFILE ステートメント

SAS データセットを開き、ファイル全体をメモリに保持するために必要なバッファを割り当てます。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

SASFILE ステートメントで開いた SAS データセットは、後続の入力(読み取り)処理や更新処理で使用されますが、出力処理やユーティリティ 処理では使用されません。

## 構文

```
SASFILE <libref.> member-name<.member-type> <(password-option(s))>  
OPEN | LOAD | CLOSE;
```

## 引数

### *libref*

SAS ライブラリに関連付けられた名前を指定します。libref (ライブラリ参照名)には、有効な SAS 名を指定する必要があります。デフォルトのライブラリ参照名は、USER(割り当てられている場合)または WORK(USER が割り当てられていない場合)になります。

**制限事項** ライブラリ参照名には、順次形式のライブラリを含む SAS ライブラリの連結を指定することはできません。

### *member-name*

ライブラリ参照名に関連付けられた SAS ライブラリのメンバを示す有効な SAS 名を指定します。

**制限事項** SAS データセットは、V7、V8、V9 SAS Engine を使用して作成する必要があります。

### *member-type*

開く SAS ファイルの種類を指定します。有効な値は DATA です。これがデフォルト設定になります。

### *password-option(s)*

次のパスワードオプションを 1 つまたは複数指定します。

**ENCRYPTKEY=key-value**

SASFILE ステートメントで AES (Advanced Encryption Standard) で暗号化された SAS データファイルを開きます。SAS データファイルが AES (Advanced Encryption Standard) アルゴリズムで暗号化されている場合、ファイルにアクセスするにはファイルに割り当てられているキー値を指定しなければなりません。キー値は最大で 64 バイト長になります。

**操作** AES 暗号化の SAS データファイルに ENCRYPTKEY=オプションを指定しない場合、ダイアログボックスが表示されキー値の指定を促します。

**READ=password**

SASFILE ステートメントで読み取り保護が設定されたファイルを開くことができます。*password* には、有効な SAS 名を指定する必要があります。

**WRITE=password**

SASFILE ステートメントで WRITE パスワードを使用して、読み取り保護と書き込み保護の両方が設定されたファイルを開くことができます。*password* には、有効な SAS 名を指定する必要があります。

**ALTER=password**

SASFILE ステートメントで ALTER パスワードを使用して、読み取り保護と変更保護の両方が設定されたファイルを開くことができます。*password* には、有効な SAS 名を指定する必要があります。

**PW=password**

SASFILE ステートメントでパスワードを使用して、すべてのレベルの保護が設定されたファイルを開くことができるようにします。*password* には、有効な SAS 名を指定する必要があります。

ヒ SASFILE ステートメントの実行時、ファイルに読み取り保護が設定されているかどうかチェックされます。そのため、ファイルに読み取り保護が設定されている場合、SASFILE ファイルステートメントに READ=パスワードを指定する必要があります。ファイルに書き込み保護または変更保護が設定されている場合、WRITE=、ALTER=、PW=の各パスワードを使用できます。ただし、ファイルは入力(読み取り)モードのみで開かれます。また、後続の処理では、必要なパスワードを1つまたは複数指定する必要があります。“例 2: SASFILE ステートメントでパスワードを指定する”(374 ページ)を参照してください。

**OPEN**

ファイルを開き、バッファを割り当てます。ただし、プロシジャ、ステートメント、アプリケーションが実行されるまでメモリへのデータの読み込みを保留します。

**LOAD**

ファイルを開き、バッファを割り当ててから、データをメモリに読み込みます。

注: 許可されているバッファの合計数がファイルに必要なバッファ数(データセットのページ数やインデックスファイルのページ数に基づいて算出)よりも少ない場合、メモリに読み込むページ数を示す警告が発行されます。

**CLOSE**

バッファを解放し、ファイルを閉じます。

## 詳細

### 概要情報

SASFILE ステートメントでは SAS データセットを開き、ファイル全体をメモリに保持するために必要なバッファを割り当てます。ファイルが読み込まれると、データはメモリに保持されます。保持されたデータは、2 番目の SASFILE ステートメントでファイルを閉じてバッファを解放するか、プログラムの終了によりファイルを自動的に閉じてバッファを解放するまで、後続の DATA ステップまたは PROC ステップで使用できます。

SASFILE ステートメントを使用すると、次の削減によってパフォーマンスが改善されます。

- SAS データを処理するために複数回開く/閉じる操作(バッファに対するメモリの割り当てや解放も含む)を、1 回の開く/閉じる操作に減らします。
- データをメモリに保持して I/O 処理を実行します。

SAS プログラムが、SAS データセットを複数回読み込むステップで構成され、ファイル全体を実メモリに保持するのに十分なメモリがある場合、プログラムで SASFILE ステートメントを使用すると便利です。ただし、ご使用の環境でテストを設定し、SASFILE ステートメント使用時と未使用時のパフォーマンスを計測することをお勧めします。

### SASFILE ステートメントで開いた SAS データセットを処理する

SASFILE ステートメントを実行すると、指定したファイルが開かれます。後続の DATA ステップまたは PROC ステップの実行時、リクエストごとにファイルを開きません。2 番目の SASFILE ステートメントでファイルを閉じるか、プログラムまたはセッションが終了するまでファイルは開いたままの状態になります。

SASFILE ステートメントで SAS データセットを開くと、ファイルは入力処理モードで開かれます。そのため、後続の入力処理や更新処理でこのファイルを使用できます。ただし、ユーティリティ処理や出力処理ではファイルに対して排他的アクセス(メンバレベルのロック)が必要になります。そのため、後続のユーティリティ処理または出力処理ではこのファイルを使用できません。たとえば、ファイルの置き換えや、変数名の変更は実行できません。

SAS プロシジャおよびステートメントの一覧と SASFILE ステートメントでファイルを開いた場合に処理が許可されるかどうかを次の表に示します。

表 2.11 SASFILE で開いたファイルに対するリクエストの処理

リクエストの処理	ファイルを開くときのモード	許可状況
APPEND プロシジャ	更新	許可
ファイルの作成や置き換えを行う DATA ステップ	出力	不許可
変数名の変更や変数の追加、ラベルの追加や変更、一貫性制約またはインデックスの追加や削除を行う DATASETS プロシジャ	ユーティリティ	不許可

リクエストの処理	ファイルを開くときのモード	許可状況
AGE、CHANGE、DELETE ステートメントを含む DATASETS プロシジャ	ファイルは開かないが、排他的アクセスが必要	いいえ
FSEDIT プロシジャ	更新	許可
PRINT プロシジャ	入力	許可
元のデータセットを並べ替え済みのデータセットに置き換える SORT プロシジャ	出力	不許可
オブザベーションの変更、追加、削除を行う SQL プロシジャ	更新	許可
CREATE TABLE または CREATE VIEW ステートメントを含む SQL プロシジャ	出力	不許可
一貫性制約またはインデックスの作成や削除を行う SQL プロシジャ	ユーティリティ	不許可

### バッファの割り当て

バッファとは、データ処理中にデータのセグメントの保持に使用する予約されたメモリ領域です。割り当てるバッファ数によって、1度にメモリに保持できるデータ量が特定されます。

バッファ数は SAS ファイルに恒久的に割り当てられる属性ではありません。つまり、バッファ数は現在の SAS セッションまたはジョブの実行中のみ有効です。SAS ファイルを開くと、ファイル処理に使用するバッファ数のデフォルト値が設定されます。デフォルト値はご使用の環境によって変わりますが、通常は小さい数字になります。特定のバッファ数を指定するには、BUFNO=システムオプションまたはデータセットオプションを使用します。

SASFILE ステートメントを実行すると、データセットのページ数やインデックスファイルのページ数(インデックスファイルが存在する場合)に基づいて、バッファ数が自動的に割り当てられます。次に例を示します。

- データセットのページ数が 5 でインデックスファイルが存在しない場合、5 バッファが割り当てられます。
- データセットのページ数が 500 でインデックスファイルのページ数が 200 の場合、700 バッファが割り当てられます。

メモリに保持されるファイルのサイズが処理中に大きくなる場合、ファイルを格納するために割り当てるバッファ数も増えます。SAS データセットに対して SASFILE ステートメントを実行する場合、BUFNO=オプションは無視されます。



## I/O 処理

I/O(入力/出力)リクエストにより、ストレージデバイス(例:ディスク)からデータのセグメントが読み込まれ、そのデータがメモリに転送されます。また、反対に、メモリからデータが転送され、そのデータがストレージデバイスに書き込まれます。SASFILE ステートメントで SAS データセットを開くと、データを 1 度読み込んでからメモリに保持するので、I/O リクエストの数を減らすことができます。

### 注意:

ただし、I/O 処理を減らすことができるのは、実メモリが十分に存在する場合のみです。SAS データセットが大きい場合、ファイル全体を保持するのに必要なだけ実メモリを割り当てられない場合があります。十分なメモリが存在しない場合、動作環境によって実在するよりも多くメモリが割り当てられます。これは仮想メモリと呼ばれます。仮想メモリが発生すると、データアクセス I/O リクエストはスワッピング I/O リクエストに置き換えられます。そのため、パフォーマンスは改善されません。また、SAS と動作環境の両方に割り当て可能な最大メモリサイズが設定されています。プログラムが必要な場合には、このサイズ以上のメモリを割り当てることができます。プログラムで使用可能なサイズ以上のメモリが必要になると、メモリを解放するため、割り当て済みのバッファ数がデフォルトのバッファ数まで減らされます。

**ヒント** SAS データセットに必要なメモリサイズを特定するには、そのファイルに対して CONTENTS プロシジャを実行し、ページサイズ、データセットのページ数、インデックスファイルのサイズ、インデックスファイルのページ数を取得します。

## 比較

特定のバッファ数を指定するには、BUFNO=システムオプションまたはデータセットオプションを使用します。

## 例

### 例 1: 複数のステップを含むプログラムで SASFILE ステートメントを使用する

次の SAS プログラムは、SAS データを開いてから、そのデータをメモリに転送し、メモリに保持されたデータを複数のタスクで読み込むプロセスを示しています。このプログラムはファイルを複数回読み込むステップで構成されています。

```
libname mydata 'SAS-library';
sasfile mydata.census.data open; 1
data test1;
  set mydata.census; 2
run;
data test2;
  set mydata.census; 3
run;
proc summary data=mydata.census print; 4
run;
data mydata.census; 5
  modify mydata.census;
.
. (statements to modify data)
.
run;
```

```
sasfile mydata.census close;6
```

- 1 SAS データセット Mydata.Census を開き、データセットのページ数とインデックスファイルのページ数に基づいてバッファ数を割り当てます。
- 2 Mydata.Census のページをすべて読み込み、すべてのデータをディスクからメモリに転送します。
- 3 Mydata.Census の 2 回目の読み込みを行います。今回は I/O リクエストを実行せず、メモリから読み込みます。
- 4 Mydata.Census の 3 回目の読み込みを行います。今回も I/O リクエストを実行せず、メモリから読み込みます。
- 5 Mydata.Census の 4 回目の読み込みを行います。今回も I/O リクエストを実行せず、メモリから読み込みます。MODIFY ステートメントによってメモリ内のデータが正常に変更されると、変更されたデータが DATA ステップの最後にメモリからディスクに転送されます。
- 6 Mydata.Census を閉じてから、割り当てたメモリを解放します。

### 例 2: SASFILE ステートメントでパスワードを指定する

次の SAS プログラムは、SASFILE ステートメントを使用し、読み取り保護と変更保護の両方が設定されている SAS データセットのパスワードを指定する例を示しています。

```
libname mydata 'SAS-library';
sasfile mydata.census (read=gizmo) open;1
proc print data=mydata.census (read=gizmo);2
run;
data mydata.census;
  modify mydata.census (alter=luke);3
.
. (statements to modify data)
.
run;
```

- 1 SASFILE ステートメントで読み取りパスワードを指定します。これは、ファイルを開くために使用します。
- 2 PRINT プロシジャでは、読み取りパスワードを再度指定する必要があります。
- 3 MODIFY ステートメントでは、データセットが更新されるので、変更パスワードを使用します。

注: この例では、READ パスワードのかわりに、権限レベルが高い ALTER パスワードを使用することができます。

### 関連項目:

#### データセットオプション:

- [“BUFNO= Data Set Option” \(SAS Viya Data Set Options: Reference\)](#)

#### システムオプション:

- [“BUFNO= System Option” \(SAS Viya System Options: Reference\)](#)

## SELECT ステートメント

複数のステートメントまたはステートメントグループからその 1 つを実行します。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
制御

**種類:** 実行

### 構文

```
SELECT <(select-expression)>;
    WHEN-1 (when-expression-1 <..., when-expression-n> ) statement;
    <... WHEN-n (when-expression-1 <...,when-expression-n> ) statement;>
    < OTHERWISE statement;>
```

END;

### 引数

#### (select-expression)

1 つの値に評価される任意の SAS 式を指定します。

参照項目: [“select-expression を指定する場合の when-expression の評価” \(376 ページ\)](#)

#### (when-expression)

複合式などの任意の SAS 式を指定します。SELECT では、when-expression を少なくとも 1 つ指定する必要があります。

**ヒント** 複数の when-expressions をカンマで区切ることは、論理演算子 OR で区切ることに相当します。

when-expression の使われ方は、select-expression を指定するかどうかによって異なります。

参照項目: [“select-expression を指定しない場合の when-expression の評価” \(376 ページ\)](#)

#### statement

DO ステートメント、SELECT ステートメント、ヌルステートメントなどの SAS 実行ステートメントを指定します。statement 引数は必ず指定する必要があります。

### 詳細

#### SELECT グループ内の WHEN ステートメントの使用

SELECT ステートメントは、SELECT グループを開始します。SELECT グループ内に WHEN ステートメントを置き、条件が真の場合に実行する SAS ステートメントを指定します。WHEN ステートメントを SELECT グループ内に少なくとも 1 つ使用する必要があります。オプションの OTHERWISE ステートメントは、

WHEN の条件が偽の場合に実行するステートメントを指定します。END ステートメントは、SELECT グループを終了します。

WHEN ステートメントにヌルステートメントを使用すると、真の場合に追加アクションを伴わない条件として認識されます。OTHERWISE ステートメントにヌルステートメントを使用すると、WHEN の条件がすべて偽の場合にエラーメッセージの表示を避けることができます。

### ***select-expression* を指定する場合の *when-expression* の評価**

*select-expression* を指定する場合、*select-expression* と *when-expression* が評価されます。この 2 つの値が等しいかどうかを比較した後、真または偽の値を返します。比較結果が真ならば、*statement* を実行します。比較結果が偽ならば、現在の WHEN ステートメント内の次の *when-expression* に移動するか、他に式が指定されていない場合は次の WHEN ステートメントに移動します。WHEN ステートメントが他に存在せず、OTHERWISE ステートメントが指定されている場合は OTHERWISE ステートメントに移動します。SELECT-WHEN の比較結果がすべて偽であり、OTHERWISE ステートメントが指定されていない場合は、エラーメッセージを表示して DATA ステップの実行を中止します。

### ***select-expression* を指定しない場合の *when-expression* の評価**

*select-expression* を指定しない場合、*when-expression* が評価されて真または偽の結果が生成されます。結果が真ならば、*statement* を実行します。結果が偽ならば、現在の WHEN ステートメント内の次の *when-expression* に移動するか、他に式が指定されていない場合は次の WHEN ステートメントを実行します。また、OTHERWISE ステートメントが指定されている場合は、OTHERWISE ステートメントに移動します。(つまり、最初の真の WHEN ステートメントに示されたアクションを実行します。) *when-expressions* の結果がすべて偽になり、OTHERWISE ステートメントが指定されていない場合は、エラーメッセージを表示します。複数の WHEN ステートメントで *when-expression* が真になる場合、最初の WHEN ステートメントが使用されます。いったん *when-expression* が真になると、それ以降の *when-expressions* は評価されません。

### **%INCLUDE ファイルを使用した大量のデータの処理**

大量のデータを処理する方法の 1 つとして、DATA ステップ内で %INCLUDE ステートメントを使用することが挙げられます。%INCLUDE ステートメントを使用すると、メインプログラムを管理しやすい状態に保ったまま複雑な処理を実行できます。メインプログラムで使用する %INCLUDE ファイルは、データを処理する WHEN ステートメントとその他の SAS ステートメントから構成されます。例については、“[例 5: 大量のデータの処理](#)” (377 ページ) を参照してください。

## **比較**

ステートメントが少ないプログラムの場合には、IF-THEN/ELSE ステートメントを使用します。IF 句に指定した条件が真になるオブザベーションやレコードのみに対して処理を継続するには、サブセット化 IF ステートメントを THEN 句を指定せずに使用します。

SELECT ステートメントは SQL プロシジャの CASE ステートメントのように動作します。

## **例**

### **例 1: ステートメントの使用**

```
select (a);
```

```

when (1) x=x*10;
when (2);
when (3,4,5) x=x*100;
otherwise;
end;

```

## 例 2: DO グループの使用

```

select (payclass);
  when ('monthly') amt=salary;
  when ('hourly')
    do;
      amt=hrlywage*min(hrs,40);
      if hrs>40 then put 'CHECK TIMECARD';
    end; /* end of do */
  otherwise put 'PROBLEM OBSERVATION';
end; /* end of select */

```

## 例 3: 複合式の使用

```

select;
  when (mon in ('JUN', 'JUL', 'AUG')
    and temp>70) put 'SUMMER ' mon=;
  when (mon in ('MAR', 'APR', 'MAY'))
    put 'SPRING ' mon=;
  otherwise put 'FALL OR WINTER ' mon=;
end;

```

## 例 4: 等しいかを比較

```

/* INCORRECT usage to select value of 2 */
select (x);
/* evaluates T/F and compares for */
/* equality with x */
when (x=2) put 'two';
end;
/* correct usage */
select(x);
/* compares 2 to x for equality */
when (2) put 'two';
end;
/* correct usage */
select;
/* compares 2 to x for equality */
when (x=2) put 'two';
end;

```

## 例 5: 大量のデータの処理

次の例の%INCLUDE ステートメントには、インベントリ内の新しい項目や古い項目を処理する WHEN ステートメントからなるコードが含まれます。メインプログラムには、DATA ステップの全体的なロジックが示されます。

```

data test (keep=ItemNumber);
  set ItemList;
  select;
    %include NewItems;
    %include OldItems;

```

```

        otherwise put 'Item ' ItemNumber ' is not in the inventory.';
    end;
run;

```

## 関連項目:

### ステートメント:

- [“DO ステートメント” \(62 ページ\)](#)
- [“IF ステートメント、サブセット化” \(134 ページ\)](#)
- [“IF-THEN/ELSE ステートメント” \(137 ページ\)](#)

---

## SET ステートメント

1 つ以上の SAS データセットからオブザベーションを読み込みます。

**該当要素:** DATA ステップ

**カテゴリ:** CAS  
ファイル操作

**種類:** 実行

**注:** SET ステートメントを使用して読み込まれた変数は PDV で保持されます。読み込まれた変数のデータタイプも保持されます。詳細については、[“RETAIN ステートメント” \(361 ページ\)](#)を参照してください。

**ヒント:** SET ステートメントを使用してデータを読み込む方法については、SAS チュートリアルビデオ [SAS データセットの読み込み](#) をご覧ください。

---

## 構文

```

SET<SAS-data-set(s) <(data-set-options(s) )> >
  <options>;

```

### 引数なし

引数を指定しない場合、SET ステートメントは最近作成されたデータセットからオブザベーションを読み込みます。

### 引数

#### *SAS-data-set (s)*

1 レベル名、2 レベル名、特殊 SAS データセット名の 1 つを指定します。

**ヒ** データセットのリストを指定できます。詳細については、[“SET ステートメントでデータセットを使用する” \(385 ページ\)](#)を参照してください。  
**ン**  
**ト**

---

データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

---

例 “例 13: データセットリストを使用する” (390 ページ)

**(data-set-options)**

処理対象の変数またはオブザベーションをプログラムデータベクトルに読み込むときに実行するアクションを指定します。

ヒント データセットリストに適用するデータセットオプションは、データセットリストに存在するすべてのデータセットに適用されます。

参照項目 入力データセットに使用できるデータセットオプションの一覧については、“[Definition of Data Set Options](#)” (*SAS Viya Data Set Options: Reference*)を参照してください。

**SET オプション**

**CUROBS=variable**

データセットから読み込んだオブザベーション数を含む変数を作成および名前を付与

例 “例 14: 現在のオブザベーション番号を調べる” (391 ページ)

**END=variable**

作成する一時変数の名前を指定します。この変数の値には終端指示子が格納されます。この変数の値は 0 に初期化されますが、SET ステートメントに指定されている最後のデータセットから最後のオブザベーションを読み込むときに 1 に設定されます。この変数は、新しいデータセットには追加されません。

制限事項 END=オプションは POINT=オプションと併用できません。ランダムアクセスを使用する場合、END=に指定した変数の値は 1 に設定されません。

操作 BY ステートメントを使用する場合、インタリーブされたデータセットから最後のオブザベーションを SET ステートメントが読み込むときに END=の値が 1 に設定されます。詳細については、“[SET を使用した BY グループ処理](#)” (386 ページ)を参照してください。

例 “例 11: すべてのオブザベーション読み込み後にオブザベーションを書き込む” (389 ページ)

**KEY=index</UNIQUE>**

インデックス変数またはキーの値に基づいて、SAS データセット内のオブザベーションに非順次アクセスします。

範囲 読み込むデータセットの単一インデックス名または複合インデックス名を指定します。

制限事項 このオプションは CAS Server ではサポートされていません。

VARCHAR データタイプの変数はサポートされていません。

KEY=オプションは POINT=オプションと併用できません。

**ヒント** SYSRC 自動呼び出しマクロと `_IORC_` 自動変数を組み合わせて使用すると、エラー処理に活用できる情報をこれまでよりも多く得ることができます。SET ステートメントに `KEY=オプション` を使用すると、自動変数 `_IORC_` が作成されます。この自動変数の値は、SAS データセットのオブザベーションに対して実行した、最新の入出力処理のステータスを示すリターンコードに設定されます。KEY=オプションの値が見つからない場合、`_IORC_` 変数は SYSRC 自動呼び出しマクロのニーマニック `_DSENUM` に対応する値を返します。また、自動変数 `_ERROR_` の値を 1 に設定します。

SET ステートメントを `KEY=オプション` と重複するインデックスで使用した場合、SET ステートメントはキー値に一致する最初のオブザベーションから読み込み開始するようにした方が良い結果が得られます。KEYRESET=オプションを使って、読み込むデータセットのインデックスの先頭から `KEY=オプション` に指定された値の検索を開始するかどうかをコントロールします。

**参照項目:** 詳細については、SAS Viya マクロ言語リファレンスの SYSRC 自動呼び出しマクロの説明を参照してください。

[“KEYRESET=variable” \(380 ページ\)](#)

[UNIQUE オプション \(383 ページ\)](#)

**例** [“例 7: テーブルルックアップの実行” \(388 ページ\)](#)

[“例 8: マスタファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する” \(388 ページ\)](#)

**注意** **KEY=オプションを使用するときは、無限ループが発生する可能性があります。** プライマリデータセットを指定せずに `KEY=オプション` を使用する場合、DATA ステップの処理を終了させる STOP ステートメント、または `_IORC_` 自動変数と SYSRC 自動呼び出しマクロを組み合わせて使用し、`_IORC_` 変数の無効な値をチェックするプログラムロジックのどちらかまたは両方を使用する必要があります。

### KEYRESET=variable

読み込むデータセットのインデックスの先頭から `KEY=オプション` に指定された値の検索を開始するかどうかをコントロールします。KEYRESET 変数の値が 1 の場合、インデックスの一番上から検索します。KEYRESET 変数の値が 0 の場合、インデックスの検索はリセットされないの、最後の検索のつづきから検索します。

**制限事項** VARCHAR データタイプの変数はサポートされていません。

**操作** KEYRESET=オプションは UNIQUE オプションに似ていますが、KEYRESET=オプションでは KEY = 検索をインデックスの最初からにするかどうかを決められます。

**参照項目:** [“KEY=index</UNIQUE>” \(379 ページ\)](#)

[“UNIQUE” \(383 ページ\)](#)

**例** [“例 15: KEYRESET オプションの使用” \(391 ページ\)](#)



**INDSNAME=variable**

作成する変数の名前を指定します。この変数には、現在のオブザベーションの読み込み先となる SAS データセット名が格納されます。格納される名前はデータセット名または物理名になります。物理名には動作環境でファイルを判別できる名前を指定します。

ヒ データセット名の場合、変数の値にライブラリ名を追加した後  
ン (例:WORK.PRICE)、この 2 レベル名を大文字に変換します。  
ト

定義されていない場合、変数の長さは 41 バイトに設定されます。ファイル名が 41 バイトより長い場合、LENGTH ステートメントを使用すると、物理ファイル名を格納できる長さまで変数の長さを拡張できます。

この変数が特定の長さを指定した文字変数として定義されている場合、変数の長さは変更されません。INDSNAME 変数の値が定義した長さを超えている場合、値は切り捨てられます。

この変数が数値変数として定義されている場合はエラーが発生します。

変数は DATA ステップのどの位置にでも指定できますが、変数は出力には追加されません。

例 “例 12: 現在のオブザベーションを読み込むデータセットの名前を取得する” (389 ページ)

**NOBS=variable**

作成する一時変数の名前を指定します。この一時変数の値は、1 つまたは複数の入力データセットの中のオブザベーションの合計数になります。SET ステートメントに複数のデータセットを指定する場合、NOBS=オプションの変数の値は、指定した複数のデータセットに含まれるオブザベーションの合計数になります。オブザベーション数には、削除対象としてマークされていても、まだ削除されていないオブザベーションも含まれます。

制限 特定の SAS ビューと TAPE や XML エンジンのような順次エンジンに  
事項 対しては、SAS がオブザベーション数を検知することができません。  
この場合、NOBS=オプションに指定した変数の値には、動作環境で使用できる最大の正の整数値が設定されます。

操作 NOBS=オプションと POINT=オプションは互いに独立しています。

ヒント コンパイル時に、各データセットのディスクリプタ情報を読み込んでから、NOBS=オプションの変数の値を自動的に割り当てます。そのため、SET ステートメントの実行前にも NOBS=オプションの変数を参照できます。この変数は DATA ステップで使用できますが、出力データセットには追加されません。

例 “例 10: 最終オブザベーションに到達するまで関数を実行する” (389 ページ)

**OPEN=(IMMEDIATE | DEFER)**

処理の準備ができるまで、連結された SAS データセットを開くことを保留できます。

**IMMEDIATE**

コンパイル中に、SET ステートメントに指定されているすべてのデータセットを開きます。

**制限事項** IMMEDIATE オプションを使用する場合、KEY=、POINT=、BY ステートメントの処理を使用できますが、使用できるのはそのうち 1 つだけです。

**ヒント** 後から開いたデータセットの変数の種類が最初に開いたデータセットにある同じ名前の変数の種類と異なる場合(例:最初の変数の種類が文字で後の変数の種類が数値など)、DATA ステップの処理は中止され、エラーメッセージが出力されます。

### DEFER

コンパイル中に最初のデータセットを開きます。後続のデータセットは実行中に開きます。1 つの DATA ステップ内のすべてのオブザベーションの読み込みと処理が終了すると、そのオブザベーションを閉じてから、指定されている次のデータセットを開きます。

**制限事項** DEFER オプションを指定する場合、KEY=ステートメントオプション、POINT=ステートメントオプション、BY ステートメントは使用できません。この 3 つによって、データセットのオブザベーションをランダムに処理するかインタリーブするかを暗示的に指定します。ただし、すべてのデータセットが開いていない場合、この処理を実行できません。

**要件** 変数の処理に DROP=、KEEP=、RENAME=データセットオプションを使用できますが、各データセットで処理する変数は同一の変数でなければなりません。通常、後続のデータセットで定義された変数が最初のデータセットで定義された変数と異なる場合、警告メッセージが出力されますが、処理は中止せずに行われます。

- 後から開いたデータセットの変数の種類が最初に開いたデータセットにある同じ名前の変数の種類と異なる場合(例:最初の変数の種類が文字で後の変数の種類が数値など)、DATA ステップの処理は中止され、エラーメッセージが出力されます。
- 後から開いたデータセットの変数が SET ステートメントで最初に開いたデータセットには定義されていないが、DATA ステッププログラムにあらかじめ定義されている場合、DATA ステップの処理は中止され、エラーメッセージが出力されます。この場合、これまでの反復処理でのこの変数の値が不適切である可能性があります。SET ステートメントのセマンティック動作では、最初のデータセットから最初のオブザベーションを処理する際にこの変数を欠損値に設定する必要があります。

デフォルト IMMEDIATE

### POINT=variable

一時変数の名前を指定します。この変数の数値によって、読み込むオブザベーションが特定されます。POINT=オプションを指定すると、SET ステートメントはランダム(ダイレクト)アクセスを使用して SAS データセットを読み込みます。

**制限事項** VARCHAR データタイプの変数はサポートされていません。

このオプションは CAS Server ではサポートされていません。

POINT=オプションは、BY ステートメント、WHERE ステートメント、WHERE=データセットオプションと併用できません。また、移送用形式のデータセット、テープまたはディスク上の順次形式のデータセット、SAS/ACCESS ビュー、外部ファイルからデータを読み込む SQL プロシジャビューと併用することもできません。

POINT=オプションと KEY=オプションを併用することはできません。

要件 STOP ステートメント

注 N は繰り返しの回数で、最後に読み込まれたオブザベーションのオブザベーション番号ではありません。

ヒント POINT=変数の値を入力しなければなりません。たとえば、DO ステートメントの一部の形式では、POINT=変数をインデックス変数として使用できます。

POINT=オプションに指定した変数は DATA ステップのどの位置にでも指定できますが、新しい SAS データセットには追加されません。

例 “例 6: 1 オブザベーションと複数オブザベーションを結合する” (388 ページ)

“例 9: ダイレクトアクセスを用いたサブセットの読み込み” (388 ページ)

注意 **POINT=オプションを使用するときは、無限ループが発生する可能性があります。** POINT=オプションを使用する場合、DATA ステップの処理を終了させる STOP ステートメント、または POINT=オプションに指定した変数の無効な値をチェックするプログラムロジックのどちらかまたは両方を使用する必要があります。POINT=オプションでは DO ステートメントに指定したオブザベーションだけを読み込むので、順次アクセスでファイルを読み込む場合とは違い、ファイル終端指示子を読み取ることができません。ファイル終端指示子を読み取ることによって DATA ステップを自動的に終了させるため、POINT=オプションの使用時に DATA ステップを終了させる別の方法を準備しないと、DATA ステップで無限ループが発生する場合があります。POINT=オプションに指定した変数の値が無効な場合、自動変数 ERROR の値が 1 に設定されます。この情報を使用すると、DO ループが無限に発生する、または STOP ステートメントが DATA ステップの最後に含まれているなどの状態をチェックできます。

## UNIQUE

読み込むデータセットのインデックスの先頭から KEY=オプションに指定された値の検索を開始します。

制限事項 UNIQUE は KEY=オプションを指定した場合にのみ指定できます。また、スラッシュを前に追加する必要があります。

注 デフォルトでは、SET ステートメントは、KEY=オプションの値が変更された場合にのみインデックスの先頭から検索を開始します。

SET ステートメントを繰り返し実行するときに KEY=の値が変更されない場合、最近取得したオブザベーションの直後から検索を開始します。

KEY=オプションに指定した変数の値の重複が連続して存在する場合、SET ステートメントは読み込み中のデータセットにある重複したインデックス付きの値に対して 1 対 1 で対応させようとします。KEY=オプションに指定した変数の値の重複が読み込み中のデータセットに存在する数よりも多い場合、それ以上の重複は検出されなかったと判断されます。

KEY=オプションに指定した変数の値が重複しない場合、最初にそのキーの値を使用してオブザベーションを読み込む場合は成功しますが、2 回目以降に同じ値を使用してオブザベーションを読み込もうとすると失敗します。この場合、\_IORC\_変数は、SYSRC 自動呼び出しマクロの二一モニック\_DSENOM に対応する値を返します。/UNIQUE オプションを追加すると、初回以降でも KEY=オプションに指定した変数の重複しない値を使用してオブザベーションを正常に読み込むことができます。この場合、\_IORC\_変数は 0 を返します。

参照項目: 詳細な例については、*Combining and Modifying SAS Data Sets: Examples* も参照してください。

["KEYRESET=variable" \(380 ページ\)](#)

例 [“例 8: マスタファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する” \(388 ページ\)](#)

## 詳細

### SET の処理

SET ステートメントを実行するたびに、1 つのオブザベーションをプログラムデータベクトルに読み込みます。他の処理を指示しない限り、SET ステートメントでは、すべての変数とすべてのオブザベーションを入力データセットから読み込みます。SET ステートメントでは複数のデータセットを使用できます。また、1 つの DATA ステップには複数の SET ステートメントを指定できます。

*Combining and Modifying SAS Data Sets: Examples* を参照してください。

注: DATA ステップがファイル終端マーカまたはすべての開かれているデータセットの終わりに到達すると、シャットダウンが行われます。たとえば、SET ステートメントを FIRSTOBS と併用すると、複数のファイルの中でヘッダレコードのみを含むファイルによって、DATA ステップの通常のシャットダウンがトリガされます。このシャットダウンは、ファイル終端指示子を越えて読み込みを行うと DATA ステップが終了するために発生します。END=オプションを使用すると、シャットダウンを回避することができます。

### 使用法

SET ステートメントはプログラミング時にさまざまな方法で柔軟に使用できます。SET ステートメントをオプションやステートメントと組み合わせることにより、次のことが行えます。

- DATA ステップでさらに処理を行うために、既存の SAS データセットからオブザベーションと変数の値を読み込みます。
- データセットを連結します。データセットをインタリーブします。また、データセットの 1 対 1 の読み込みを実行します。
- ダイレクトアクセス方式を使用して SAS データセットを読み込みます。

## SET ステートメントでデータセットを使用する

SET ステートメントでは、データセットリストを使用できます。データセットリストを使用すると、現在存在するデータセットのグループを簡単に示せます。このデータセットリストには、名前接頭辞リストまたは番号付き範囲リストを指定する必要があります。

**名前接頭辞リスト**は、指定した文字列で始まるすべてのデータセットを示します。たとえば、**set SALES1;**と指定すると、"SALES1"で始まる SALES1、SALES10、SALES11、SALES12 などのデータセットがすべて読み込まれます。

**番号付き範囲リスト**では、連番である最後の文字を除き、同じ名前のデータセットが存在する必要があります。番号付き範囲リストでは、開始値と終了値に任意の数値を使用してもかまいません。たとえば、次のリストは同じデータセットを示します。

```
sales1 sales2 sales3 sales4
sales1-sales4
```

**注:** 最初のデータセット名にある数値接尾辞の先頭に 0 が置かれる場合、最後のデータセット名の数値接尾辞の桁数は最初のデータセット名の桁数と一致するか、または上回る必要があります。このように指定されていない場合はエラーが発生します。たとえば、データセットリストを sales001-sales99 または sales01-sales9 と指定するとエラーが発生します。ただし、このデータセットリストの有効な指定は、sales001-sales999 です。最初のデータセット名にある数値接尾辞の先頭に 0 が置かれない場合、最初と最後のデータセット名に使用する数値接尾辞の桁数を一致させる必要はありません。たとえば、データセットリストの指定として sales1-sales999 は有効です。

番号付きデータセットリストを使用する場合に考慮すべきその他のルールを次に示します。

- 範囲のグループを複数指定できます。
 

```
set cost1-cost4 cost11-cost14 cost21-cost24;
```
- 番号付き範囲リストと名前接頭辞リストを組み合わせることで指定できます。
 

```
set cost1-cost4 cost2: cost33-37;
```
- 個々のデータセットとデータセットリストを組み合わせることで指定できます。
 

```
set cost1 cost10-cost20 cost30;
```
- データセットリストを囲んだ引用符は無視されます。
 

```
/* these two lines are the same */
set sales1 - sales4;
set 'sales1'n - 'sales4'n;
```
- データセット名の空白は無効です。引用符を使用する場合、末尾にある空白は無視されます。
 

```
/* blanks in these statements will cause errors */
set sales 1 - sales 4;
set 'sales 1'n - 'sales 4'n;
/* trailing blanks in this statement will be ignored */
set 'sales1 'n - 'sales4 'n;
```
- 数値接尾辞に使用できる最大数は、2147483647 です。
 

```
/* this suffix will cause an error */
set prod2000000000-prod2934850239;
```

### SET を使用した BY グループ処理

DATA ステップでは、1 つの BY ステートメントのみ、SET ステートメントに指定できます。BY ステートメントは適用する SET ステートメントの直後に指定する必要があります。SET ステートメントにリストされるデータセットは、BY ステートメントにリストされる変数の値に基づいて並べ替えられているか、適切なインデックスが含まれている必要があります。SET ステートメントは BY ステートメントと使用すると、複数のデータセットをインタリーブします。新しいデータセットのオブザベーションは 1 つまたは複数の BY 変数の値に従って並べられます。BY グループ内は、元のデータセットの順序に従って並べられます。SET ステートメントでの BY グループの処理の例については、“[例 2: SAS データセットのインタリーブ](#)” (387 ページ) を参照してください。

### SAS データセットの結合

1 つの SET ステートメントに複数のデータセットを指定すると、指定したデータセットが連結されます。つまり、新しいデータセットのオブザベーション数は、指定した複数のデータセットにあるオブザベーション数の合計になります。オブザベーションは、最初のデータセットのすべてのオブザベーションの後ろに、2 番目のデータセットのすべてのオブザベーションが続くという順序で配置されます。データセットの連結の例については、“[例 1: SAS データセットの連結](#)” (387 ページ) を参照してください。

1 つの SET ステートメントを BY ステートメントとともに使用すると、指定したデータセットがインタリーブされます。新しいデータセットのオブザベーションは 1 つまたは複数の BY 変数の値に従って並べられます。BY グループ内は、元のデータセットの順序に従って並べられます。データセットのインタリーブの例については、“[例 2: SAS データセットのインタリーブ](#)” (387 ページ) を参照してください。

複数の SET ステートメントを使用すると、指定したデータセットに対して 1 対 1 の読み込み(1 対 1 マッチともいう)を行います。新しいデータセットには、すべての入力データセットの変数が格納されます。新しいデータセットに含まれるオブザベーションの数は、元のデータセットのうち最も小さなデータセットに含まれるオブザベーションの数になります。複数のデータセットに共通する変数が含まれている場合、前のデータセットから読み込んだ値を最後のデータセットから読み込んだ値で置き換えます。データセットの 1 対 1 の読み込みの例は、次を参照してください。

- “[例 6: 1 オブザベーションと複数オブザベーションを結合する](#)” (388 ページ)
- “[例 7: テーブルルックアップの実行](#)” (388 ページ)
- “[例 8: マスタファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する](#)” (388 ページ)

詳細な例については、*Combining and Modifying SAS Data Sets: Examples* も参照してください。

### 比較

- SET ステートメントでは、既存の SAS データセットからオブザベーションを読み込みます。INPUT ステートメントでは、SAS 変数とオブザベーションを作成するために、外部ファイルまたはインストリームデータ行から生データを読み込みます。
- SET ステートメントで KEY=オプションを使用すると、インデックスの値に基づいて、SAS データセットのオブザベーションに非順次アクセスできます。SET ステートメントで POINT=オプションを使用すると、オブザベーションの番号に基づいて、SAS データセットのオブザベーションに非順次アクセスできます。



## 例

### 例 1: SAS データセットの連結

SET ステートメントに複数のデータセット名を指定すると、ステートメント実行後には、指定したすべてのデータセットを連結した出力データセットが作成されます。最初のデータセットからすべてのオブザベーションを読み込んだ後、2 番目のデータセットからすべてのオブザベーションを読み込むというように、すべてのデータセットからすべてのオブザベーションが読み込まれます。この例では、3 つのデータセットを連結して、FITNESS という 1 つの出力データセットを作成します。

```
data fitness;
  set health exercise well;
run;
```

### 例 2: SAS データセットのインタリーブ

2 つ以上の SAS データセットをインタリーブするには、SET ステートメントの後ろに BY ステートメントを使用します。

```
data april;
  set payable recvable;
  by account;
run;
```

### 例 3: SAS データセットの読み込み

次の DATA ステップでは、データセット NC.MEMBERS にある各オブザベーションをプログラムデータベクトルに読み込みます。次に、CITY の値が **Raleigh** であるオブザベーションのみを新しいデータセット RALEIGH.MEMBERS に出力します。

```
data raleigh.members;
  set nc.members;
  if city='Raleigh';
run;
```

### 例 4: 1 つのオブザベーションを SAS データセットのすべてのオブザベーションとマージする

既存のデータセットにマージできるオブザベーションは、SAS プロシジャまたは他の DATA ステップで作成されたオブザベーションになります。この例では、データセット AVGSALES にはオブザベーションが 1 つだけ含まれます。

```
data national;
  if _n_=1 then set avgsales;
  set totsales;
run;
```

### 例 5: 同一データセットを複数回読み込む

この例の SET ステートメントは、個別に処理されます。つまり、1 つのデータセットからの読み込みを、あたかも 2 つのデータセットからの読み込みのように処理されます。

```
data drugxyz;
  set trial5(keep=sample);
  if sample>2;
  set trial5;
```

```
run;
```

DATA ステップの反復時、最初の SET ステートメントからオブザベーションが 1 つ読み込みます。最初の SET ステートメントの実行後、次のオブザベーションが読み込まれます。DATA ステップの同じ反復内で、それぞれの SET ステートメントから別のオブザベーションが読み込まれます。

### 例 6: 1 オブザベーションと複数オブザベーションを結合する

次に示すように、ダイレクトアクセス方式を使用すると、1 つのデータセットから読み込んだ複数のオブザベーションをサブセットし、それを他のデータセットから読み込んだオブザベーションと結合できます。

```
data south;
  set revenue;
  if region=4;
  set expense point=_n_;
run;
```

### 例 7: テーブルルックアップの実行

この例では、KEY=オプションを使用して、テーブルルックアップを実行します。この DATA ステップでは、INVTORY という名前のプライマリデータセットと、PARTCODE という名前のルックアップデータセットを読み込みます。インデックス PARTNO を使用し、それぞれのデータセットで変数 PARTNO に一致する値を探しながら、PARTCODE を非順次で読み込みます。これは、プライマリデータセット INVTORY に登録されているすべてのパーツについて、ルックアップデータセット PARTCODE の変数 DESC のみに含まれる適切な説明を読み込むのが目的です。

```
data combine;
  set invtory(keep=partno instock price);
  set partcode(keep=partno desc) key=partno;
run;
```

### 例 8: マスタファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する

この例でも、KEY=オプションを使用して、テーブルルックアップを実行します。この DATA ステップでは、インデックス PARTNO が格納された INVTORY という名前のプライマリデータセットと、PARTCODE という名前のルックアップデータセットを読み込みます。PARTCODE の変数 NEW\_STK には、新しい在庫量が含まれます。UNIQUE オプションを使用することにより、INVTORY に重複する値を持つ変数 PARTNO が存在する場合、重複する値を持つグループの最初のオブザベーションにのみ NEW\_STK の値を追加します。

```
data combine;
  set partcode(keep=partno new_stk);
  set invtory(keep=partno instock price)
  key=partno/unique;
  instock=instock+new_stk;
run;
```

### 例 9: ダイレクトアクセスを用いたサブセットの読み込み

次のステートメントでは、オブザベーション番号を指定してダイレクトアクセスを行う POINT=オプションを使用し、データセット DRUGTEST から 50 オブザベーションのサブセットを選択します。

```
data sample;
```



```

do obsnum=1 to 100 by 2;
  set drugtest point=obsnum;
  if _error_ then abort;
  output;
end;
stop;
run;

```

### 例 10: 最終オブザベーションに到達するまで関数を実行する

次のステートメントでは、NOBS=オプションを使用し、DO ループ処理を終了させる値を設定します。一時変数 LAST の値は、SURVEY1 および SURVEY2 のオブザベーション数の合計になります。

```

do obsnum=1 to last by 100;
  set survey1 survey2 point=obsnum nobs=last;
  output;
end;
stop;

```

### 例 11: すべてのオブザベーション読み込み後にオブザベーションを書き込む

この例では、END=変数 LAST を使用して変数 REVENUE に値を割り当てるように指示します。また、RENTAL の最後のオブザベーションの読み込みを終えてから、オブザベーションを出力します。

```

set rental end=last;
totdays + days;
if last then
  do;
    revenue=totdays*65.78;
    output;
  end;

```

### 例 12: 現在のオブザベーションを読み込むデータセットの名前を取得する

この例では、3つのデータセットを作成し、*dsn* という名前の変数にデータセット名を格納します。名前は3つの部分に分割され、次の結果が出力されます。

```

/* Create some data sets to read */
data gas_price_option; value=395; run;
data gas_rbid_option; value=840; run;
data gas_price_forward; value=275; run;
/* Create a data set D */
data d;
  set gas_price_option gas_rbid_option gas_price_forward indsname=dsn;
  /* split the data set names into 3 parts */
  commodity = scan (dsn, 2, ".");
  type = scan (dsn, 3, ".");
  instrument = scan (dsn, 4, ".");
run;
proc print data=d;
run;

```

**アウトプット 2.30** 3つの部分に分割されたデータセット名

Obs	value	commodity	type	instrument
1	395	GAS	PRICE	OPTION
2	840	GAS	RBID	OPTION
3	275	GAS	PRICE	FORWARD

**例 13: データセットリストを使用する**

この例では、番号付き範囲リストを使用してデータセットを読み込みます。

```
data dept008; emp=13; run;
data dept009; emp=9; run;
data dept010; emp=4; run;
data dept011; emp=33; run;
data _null_;
  set dept008-dept010;
  put _all_;
run;
```

次の行が SAS ログに書き出されます。

**ログ 2.3** SET ステートメントでデータセットリストを使用する

```
1 data dept008; emp=13; run; NOTE: The data set WORK.DEPT008 has 1 observations and 1
variables.NOTE: DATA statement used (Total process time): real time 0.06 seconds cpu time 0.03
seconds 2 data dept009; emp=9; run; NOTE: The data set WORK.DEPT009 has 1 observations and 1
variables.NOTE: DATA statement used (Total process time): real time 0.00 seconds cpu time 0.00
seconds 3 data dept010; emp=4; run; NOTE: The data set WORK.DEPT010 has 1 observations and 1
variables.NOTE: DATA statement used (Total process time): real time 0.00 seconds cpu time 0.00
seconds 4 data dept011; emp=33; run; NOTE: The data set WORK.DEPT011 has 1 observations and 1
variables.NOTE: DATA statement used (Total process time): real time 0.00 seconds cpu time 0.00
seconds 5 6 data _null_; 7 set dept008-dept010; 8 put _all_; 9 run; emp=13 _ERROR_=0 _N_=1 emp=9
_ERROR_=0 _N_=2 emp=4 _ERROR_=0 _N_=3 NOTE: There were 1 observations read from the data set
WORK.DEPT008.NOTE: There were 1 observations read from the data set WORK.DEPT009.NOTE: There
were 1 observations read from the data set WORK.DEPT010.NOTE: DATA statement used (Total process
time): real time 0.00 seconds cpu time 0.00 seconds
```

加えて、欠損しているデータセットを探すのにデータセットリストを使用することもできます。この例では、番号付き範囲リストを使用して欠損しているデータセットを探します。存在しないデータセットごとにエラーが発生します。どのデータセットが欠損しているかがわかったら、実際に存在するデータセットを指定するように SET ステートメントを修正します。

```
data dept008; emp=13; run;
data dept009; emp=9; run;
data dept011; emp=4; run;
data dept014; emp=33; run;
data _null_;
  set dept008-dept014;
  put _all_;
run;
```

次の行が SAS ログに書き出されます。

**ログ 2.4** SET ステートメントを使用した存在しないデータセットの検出

```
1 data dept008; emp=13; run; NOTE: The data set WORK.DEPT008 has 1 observations and 1
variables.NOTE: DATA statement used (Total process time): real time 0.04 seconds cpu time 0.04
seconds 2 data dept009; emp=9; run; NOTE: The data set WORK.DEPT009 has 1 observations and 1
variables.NOTE: DATA statement used (Total process time): real time 0.00 seconds cpu time 0.00
seconds 3 data dept011; emp=4; run; NOTE: The data set WORK.DEPT011 has 1 observations and 1
variables.NOTE: DATA statement used (Total process time): real time 0.03 seconds cpu time 0.01
seconds 4 data dept014; emp=33; run; NOTE: The data set WORK.DEPT014 has 1 observations and 1
variables.NOTE: DATA statement used (Total process time): real time 0.00 seconds cpu time 0.00
seconds 5 data _null_; 6 set dept008-dept014; ERROR: File WORK.DEPT010.DATA does not exist.ERROR:
File WORK.DEPT012.DATA does not exist.ERROR: File WORK.DEPT013.DATA does not exist.7 put _all_; 8
run; NOTE: The SAS System stopped processing this step because of errors.NOTE: DATA statement used
(Total process time): real time 0.00 seconds cpu time 0.00 seconds
```

**例 14: 現在のオブザベーション番号を調べる**

次の例では、CUROBS オプションを使って現在のオブザベーションの番号を返すようにしています。

```
data women;
  set sashelp.class curobs=cobs;
  where sex = 'F';
  orig_obs = cobs;
run;
```

**例 15: KEYRESET オプションの使用**

この例では、KEYRESET=オプションを使って、I=3 が 2 回の時にすべての値を検索するようにしています。

```
data a(index=(i));
  do i = 1,2,3,3,3,4,5;
    j=ranuni(4);
    output;
  end;
run;
data _null_;
  input i;
  reset = 1;
  do while (_iorc_ = 0);
    set a key=i keyreset=reset;
    put _all_;
  end;
  _error_ = 0; _iorc_ = 0;
  datalines;
3
3
;
```

**関連項目:**

- *Combining and Modifying SAS Data Sets: Examples*
- [“Definition of Data Set Options” \(SAS Viya Data Set Options: Reference\)](#)
- SAS Viya マクロ言語リファレンス

**ステートメント:**

- [“BY ステートメント” \(33 ページ\)](#)

- [“DO ステートメント” \(62 ページ\)](#)
- [“INPUT ステートメント” \(179 ページ\)](#)
- [“MERGE ステートメント” \(280 ページ\)](#)
- [“STOP ステートメント” \(393 ページ\)](#)
- [“UPDATE ステートメント” \(408 ページ\)](#)

---

## SKIP ステートメント

SAS ログ内に空白行を 1 行作成します。

**該当要素:** 任意の場所

**カテゴリ:** ログ制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。

---

### 構文

**SKIP** <*n*>;

### 引数なし

引数を指定せずに SKIP を使用すると、ログ内に空白行を 1 行作成します。

### 引数

*n*

ログ内に作成する空白行の数を指定します。

**ヒント** 指定した数がページの残りの行数よりも大きい場合、次のページの最初に移動します。

---

### 詳細

SKIP ステートメント自身はログ内に表示されません。このステートメントは、操作に関するすべての方法で使用できます。

### 関連項目:

#### ステートメント:

- [“PAGE ステートメント” \(317 ページ\)](#)

#### システムオプション:

- [“LINESIZE= System Option” \(SAS Viya System Options: Reference\)](#)
- [“PAGESIZE= System Option” \(SAS Viya System Options: Reference\)](#)

---

## STOP ステートメント

現在の DATA ステップの実行を中止します。

**該当要素:** DATA ステップ

**カテゴリ:** アクション  
CAS

**種類:** 実行

---

### 構文

**STOP;**

### 引数なし

STOP ステートメントでは、現在の DATA ステップの処理を中止し、現在の DATA ステップ以降のステートメントから処理を再開します。

### 詳細

現在の DATA ステップのデータセットが書き出されます。しかし、STOP の実行時に処理されていたオブザベーションは追加されません。STOP ステートメントはそれだけでも、IF-THEN ステートメントまたは SELECT グループと組み合わせても使用できます。

STOP ステートメントは、SET ステートメント内の POINT=オプションなど、ランダムアクセスを使用して SAS データセットを読み込む機能と一緒に使用します。ランダムアクセスでは、EOFを検出できません。そのため、DATA ステップの無限ループを回避するプログラムステートメントを指定する必要があります。

### 比較

- SAS Studio または他の対話型モードの操作を使用している場合、ABORT ステートメントおよび STOP ステートメントの両方で処理を中止することができます。ABORT ステートメントでは自動変数\_ERROR\_の値を 1 に設定しますが、STOP ステートメントでは設定しません。
- バッチモードまたは非対話型モードでは、この 2 つのステートメントの結果は異なります。バッチモードまたは非対話型モードでは STOP ステートメントを使用して、処理を次の DATA ステップまたは PROC ステップから再開します。

### 例

#### 例 1: 基本的な使用

- stop;
- if idcode=9999 then stop;
- select (a);  
  when (0) output;  
  otherwise stop;  
end;

**例 2: 無限ループの回避**

この例は、ランダムアクセスを使用している場合に、STOP を使用して DATA ステップでの無限ループを回避する方法を示しています。

```
data sample;
  do sampleobs=1 to totalobs by 10;
    set master.research point=sampleobs nobs=totalobs;
    output;
  end;
  stop;
run;
```

**関連項目:****ステートメント:**

- [“ABORT ステートメント” \(17 ページ\)](#)
- [SET ステートメントの POINT=オプション \(382 ページ\)](#)

---

**合計ステートメント**

式の計算結果を合計変数に加算します。

<b>該当要素:</b>	DATA ステップ
<b>カテゴリ:</b>	アクション CAS
<b>種類:</b>	実行

---

**構文**

*variable*+*expression*;

**引数*****variable***

数値が含まれる合計変数の名前を指定します。

**ヒ** この変数の値は、最初のオブザベーションを読み込む前に自動的に 0 に  
**ン** 設定されます。RETAIN ステートメントと同じように、合計変数の値は  
**ト** 繰り返しの際に保持されます。

合計変数を 0 以外の値に初期化するには、この変数を RETAIN ステートメントで使用し、初期値を割り当てます。

---

***expression***

SAS 式を指定します。

**ヒント** 式を計算し、結果を合計変数に加算します。

式の計算結果が欠損値になる場合、0 として扱われます。

---

## 比較

合計ステートメントは、SUM 関数と RETAIN ステートメントを次のように使用することと同じです。

```
retain variable 0;
variable=sum(variable,expression);
```

## 例: 合計ステートメントの使用

さまざまな式を使用する合計ステートメントの例を次に示します。

- balance+(-debit);
- sumxsq+x\*x;
- nx+(x ne .);
- if status='ready' then OK+1;

## 関連項目:

### 関数:

- ["SUM Function" \(SAS Viya Functions and CALL Routines: Reference\)](#)

### ステートメント:

- ["RETAIN ステートメント" \(361 ページ\)](#)

---

## SYSECHO ステートメント

グローバルステートメントの完了イベントを送信し、テキスト文字列を IOM クライアントに戻します。

**該当要素:** 任意の場所

**カテゴリ:** プログラム制御

**制限事項:** このステートメントは CAS Server ではサポートされていません。  
オブジェクトサーバーモードでのみ有効

## 構文

```
SYSECHO <"text">;
```

### 引数なし

引数を指定せずに SYSECHO を使用すると、グローバルステートメントの完了イベントを IOM クライアントに送信します。

### 引数

**"text"**

IOM クライアントに戻す文字列を指定します。

**範囲** 1-64 個の文字

---

要件 テキスト文字列は二重引用符で囲む必要があります。

## 詳細

SYSECHO ステートメントを使用すると、サブミットされた SAS プログラムのセグメントの進行状況を IOM クライアントで手動でトラックできるようになります。

SYSECHO ステートメントを実行すると、グローバルステートメントの完了イベントが生成されます。また、テキストを指定した場合は、指定したテキスト文字列が IOM クライアントに戻されます。

---

## SYSTASK ステートメント

非同期タスクを一覧表示します。

**該当要素:** 任意の場所

**カテゴリ:** 動作環境

**制限事項:** このステートメントは CAS Server ではサポートされていません。

## 構文

```
SYSTASK COMMAND "operating-environment-command" <WAIT | NOWAIT>
<TASKNAME=taskname> <MNAME=name-variable> <STATUS=status-variable>
<SHELL=<="shell-command"> > <CLEANUP>;
```

```
SYSTASK LIST <_ALL_ | taskname> <STATE> <STATVAR>;
```

```
SYSTASK KILL taskname <taskname...>;
```

## 必須引数

### COMMAND

*operating-environment-command* を実行します。

### LIST

特定のアクティブなタスク、またはシステム内のすべてのアクティブなタスクをリストします。

### KILL

タスクを強制的に停止します。

### *operating-environment-command*

Linux コマンドの名前 (コマンド固有のオプションを含む)、または X Window System のアプリケーション名を指定します。一重または二重の引用符でコマンドを囲みます。コマンド固有のオプションに引用符が必要な場合は、各オプションでその引用符をくりかえします。次に例を示します。

```
SYSTASK COMMAND "xdialog -m ""There was an error."" -t ""Error"" -o";
```

コマンド名がシェルのエイリアスであるか、シェルの特殊文字であるティルダ (~) やアスタリスク (\*) をコマンド内のパス名に使っている場合、SHELL オプションを指定して、シェルがエイリアスや特殊文字を適切に処理できるようにしてください。

```
SYSTASK COMMAND "mv ~usr/file.txt /tmp/file.txt" shell;
```



この例では、SHELL オプションを指定しているので、`~usr` というパスを、そのままではなく、実行時に展開して使っています。ユーザーが指定した *operating-environment-command* は、キーボードからの入力を必須にできません。SHELL オプションが使用される場合であってもエラー内でシェルのエイリアスの結果を使用する場合は、そのシェルはシェル初期化ファイルを処理しません。SHELL エイリアスのかわりに実際の SHELL コマンドを使用します。

## オプション引数

### WAIT | NOWAIT

SYSTASK COMMAND コマンドが現在の SAS セッションの実行をタスクが完了するまで中断するかどうかを決定します。NOWAIT は初期値です。

NOWAIT オプションで始まるタスクの場合、SAS タスクが完了するまでセッションの実行を中断する必要があるときは、WAITFOR ステートメントを使用できます。

詳細については、“[WAITFOR ステートメント](#)” (415 ページ)を参照してください。

### TASKNAME=*taskname*

タスクを特定する名前を指定します。タスク名は、すべてのアクティブなタスクの中で一意である必要があります。タスクがアクティブであるとは、動作している状態、または動作を終了し、WAITFOR ステートメントで待機していない状態を言います。タスク名が重複すると、SAS ログ内でエラーが発生します。タスク名を指定しない場合は、SYSTASK が名前を自動生成します。タスク名に空白文字が含まれている場合は、タスク名全体を引用符で囲んでください。

タスクが完了しても、そのタスクに WAITFOR ステートメントを発行するか CLEANUP オプションを指定しない限り、タスク名をくりかえし使用することはできません。

### MNAME=*name-variable*

タスクに対して自動的に生成されたタスク名を SYSTASK を使って保存するための、マクロ変数を指定します。TASKNAME オプションと MNAME オプションを両方指定する場合、SYSTASK は、ユーザーが TASKNAME で指定した名前を、MNAME で指定した変数にコピーします。

### STATUS=*status-variable*

タスクのステータスを SYSTASK を使って保存するための、マクロ変数を指定します。ステータス変数は、すべてのアクティブなタスクの中で一意である必要があります。

### SHELL<=*"shell-command"*

*operating-environment-command* がオペレーティングシステムシェルコマンドを使って実行されるように指定します。*shell-command* を指定すると、SYSTASK はこのシェルコマンドを使ってシェルを起動するようになります。それ以外の場合は SYSTASK デフォルトのシェルを使用します。シェルコマンドを引用符で囲みます。

#### 動作環境の情報

シェルは *operating-environment-command* に含まれる特殊文字を展開します。

注: SHELL オプションは、ユーザーが指定した SHELL コマンドが *i* オプションを使用してステートメントを通すことを前提としています。ユーザーのシェルコマンドは、通常 **sh**、**cs****h**、**ksh**、**bash** のいずれかです。

## CLEANUP

タスクが完了したら LISTTASK 出力からタスクが削除されるよう指定します。タスクが削除されれば、WAITFOR ステートメントを発行しなくてもタスク名を再使用できます。

SYSTASK コマンドをくりかえし実行し、長時間動作し続けるジョブがある場合は、SYSTASK コマンドに WAITFOR ステートメントまたは CLEANUP オプションを指定して、メモリをクリアしてください。WAITFOR ステートメントは、SYSTASK コマンドによって開始されたすべての完了プロセスの情報を削除することによってメモリを解放します。CLEANUP オプションは、特定のジョブが完了したときにメモリをクリアし、今後の使用に備えてメモリを開放します。ジョブが完了した後に WAITFOR ステートメントを使用する場合、そのジョブはすでに CLEANUP オプションによってクリーンアップされているので、このステートメントは無効になります。

## 詳細

### 概要情報

SYSTASK を使うと、SAS セッションやアプリケーション内から、オペレーティングシステム固有のコマンドを実行できます。X ステートメントと違い、非同期タスクとして実行するので、同時に動作している他のタスクとは独立に動きまゝ。非同期タスクはバックグラウンドで実行するため、ユーザーは非同期タスクがまだ実行中であっても追加のタスクを実行できます。

### ～に固有の情報 Linux

たとえば、新しいシェルを起動し、そのシェルで Linux **cp** コマンドを実行するには次のステートメントを使用します。

```
systask command "cp /tmp/sas* ~/archive/" taskname="copyjob1"
                status=copysts1 shell;
```

**cp** コマンドからのリターンコードは、マクロ変数 **COPYSTS1** 内に保存されます。

コマンドの出力は SAS ログに表示されます。

Linux と PC では構文が違うことがあるので、PC SAS ジョブを変換して Linux 上で実行しようとしても、変換処理でエラーが発生するかも知れません。例えば次のコマンドを入力するとエラーになります。

```
systask command "md directory-name" taskname="mytask";
```

エラーになるのは、**md** が PC では「make directory」コマンドであるのに対し、Linux では何の意味もないからです。変換処理の結果、**md** は **mkdir** になります。SYSTASK ステートメントに SHELL オプションを指定しなければなりません。**mkdir** は Linux シェルに組み込みであるのに対し、PC では、**mkdir** が独立したコマンドでないからです。

ログにはエラーメッセージが書き込まれます。

注: SAS アプリケーションの SYSTASK ステートメントに続くプログラムステップは、通常は SYSTASK ステートメントの実行が成功するかどうか依存します。したがって、SYSTASK ステートメントに構文エラーがあると、SAS アプリケーションが異常終了するおそれがあります。

SAS から開始できる非同期プロセスは 2 つのタイプがあります。

### Task

SYSTASK コマンドで開始されるタスクはすべて Task タイプです。これらのタスクについて STATVAR または STATE を指定しない場合、SYSTASK LIST は、タスクの名前、タイプ、状態およびステータスのマクロ変数の名前を表

示します。SYSTASK KILL を使用すれば、Task タイプのタスクのみを終了できます。

#### SAS/CONNECT プロセス

SIGNON ステートメントまたはコマンドと RSUBMIT ステートメントを使用して SAS/CONNECT から開始されるタスクは、タイプが SAS/CONNECT プロセスです。SAS/CONNECT プロセスを表示するには、LISTTASK ステートメントを使用してタスク名、タイプおよび状態を表示します。SAS/CONNECT プロセスを終了するには、KILLTASK ステートメントを使用します。

SAS/CONNECT プロセスの詳細については、*SAS/CONNECT for SAS Viya: User's Guide* を参照してください。

注: タスク(SAS/CONNECT プロセスに限らない)を表示する望ましい方法は、SYSTASK LIST のかわりに LISTTASK ステートメントを使用することです。タスクを編集する望ましい方法は、SYSTASK KILL のかわりに KILLTASK ステートメントを使用することです。

SYSRC マクロ変数には、SYSTASK ステートメントからのリターンコードが含まれます。STATUS オプションで指定したステータス変数には、SYSTASK COMMAND で開始されたプロセスのコードが含まれています。タスクが正しく実行されるようにするには、SYSTASK ステートメントのステータスと SYSTASK ステートメントによって開始されるプロセスのステータスの、両方を監視する必要があります。

SYSTASK ステートメントの実行が失敗する場合、SYSRC マクロ変数に非ゼロ値が含まれていることがあります。たとえば、タスクを完了するためのリソースが十分でないか、SYSTASK ステートメントに構文エラーが含まれている可能性があります。SYSTASK KILL ステートメントを使用する場合、1 つ以上のプロセスが終了できなくても、SYSRC が非ゼロ値に設定されます。

タスクが開始されると、そのステータス変数が NULL に設定されます。各タスクのステータス変数を使用して、どのタスクが完了できなかったかを判断できます。ステータス変数が NULL であるタスクは実行が完了していません。タスクが異常終了すると、そのタスクのステータス変数は-1 に設定されます。ステータス変数の詳細については、“[WAITFOR ステートメント](#)” (415 ページ)を参照してください。

X ステートメントとは異なり、SYSTASK ステートメントを使用して対話セッションを新規に開始することはできません。

### 関連項目:

#### ステートメント:

- “[WAITFOR ステートメント](#)” (415 ページ)
- “[X ステートメント](#)” (423 ページ)

#### その他の参照資料:

- “[Executing Operating System Commands from Your SAS Session](#)” (*Batch and Line Mode Processing in SAS Viya*)

---

## TITLE ステートメント

SAS 出力に使用するタイトルを指定します。

該当要素: 任意の場所

<b>カテゴリ:</b>	出力制御
<b>制限事項:</b>	このステートメントは CAS Server ではサポートされていません。 TITLE ステートメントは、Unicode をサポートしていません。
<b>動作環境:</b>	タイトルの最大長

## 構文

```
TITLE <n> <ods-format-options> <'text' | "text">;
```

### 引数なし

引数を指定せずに TITLE ステートメントを使用すると、現在設定されているタイトルをすべて取り消します。

### 引数

*n*

タイトルを表示する相対行番号を指定します。

範囲 1-10

**ヒント** 番号が最大のタイトル行は、一番下の行に表示されます。*n* の指定を省略すると、この値は 1 と見なされます。そのため、最初のタイトル行には TITLE または TITLE1 を指定できます。

テキスト行の行間に空白行を追加したタイトルを作成できます。たとえば、TITLE ステートメントおよび TITLE3 ステートメントを使用してテキストを指定すると、この 2 行のテキストの間に空白行が 1 行表示されます。

### **ods-format-options**

ODS HTML、RTF、および PRINTER の各出力先に対して出力形式オプションを指定します。

#### **BOLD**

タイトルテキストを太字で表示するように指定します。

ODS 出力先 HTML、RTF、PRINTER

#### **COLOR=color**

タイトルテキストの色を指定します。

別名 C

ODS 出力先 HTML、RTF、PRINTER

例 “例 3: ODS (Output Delivery System)を使用したタイトルとフットノートのカスタマイズ” (405 ページ)

#### **BCOLOR=color**

タイトル表示部分の背景色を指定します。

ODS 出力先 HTML、RTF、PRINTER

**FONT=*font-face***

使用するフォントを指定します。複数のフォントを指定した場合、出力先デバイスはシステムにインストールされている最初のフォントを使用します。

別名 F

ODS 出力先 HTML、RTF、PRINTER

**HEIGHT=*dimension* | *size***

タイトルのフォントのサイズを指定します。

***dimension***

は正の数になります。

**ディメンジョンの測定単位**

cm センチメートル

em タイプセッティングにおける幅の標準測定単位

ex タイプセッティングにおける高さの標準測定単位

in インチ

mm ミリメートル

pt (プリンタの)ポイント

制限事項 *dimension* を指定する場合は、測定単位も指定してください。測定単位なしでは、数字は相対的なサイズになります。

***size***

*size* の値は、HTML 文書のその他すべてのフォントサイズに対して相対的です。

範囲 1 から 7

別名 H

ODS 出力先 HTML、RTF、PRINTER

例 “例 3: ODS (Output Delivery System)を使用したタイトルとフットノートのカスタマイズ” (405 ページ)

**ITALIC**

タイトルテキストをイタリック体で表示するように指定します。

ODS 出力先 HTML、RTF、PRINTER

**JUSTIFY= CENTER | LEFT | RIGHT**

配置する位置を指定します。

**CENTER**

中央揃えで表示するように指定します。

別名 C

**LEFT**

左寄せで表示するように指定します。

別名 L

### RIGHT

右寄せで表示するように指定します。

別名 R

別名 J

ODS 出力先 HTML、RTF、PRINTER

例 “例 3: ODS (Output Delivery System)を使用したタイトルとフットノートのカスタマイズ” (405 ページ)

### LINK='url'

ハイパーリンクを指定します。

ODS 出力先 HTML、RTF、PRINTER

ヒント LINK=の表示プロパティは、常に現在のスタイルから取得します。

### UNDERLIN= 0 | 1 | 2 | 3

後ろに指定するテキストに下線を表示するかどうかを指定します。0 を指定すると下線は表示されません。1、2、3 を指定すると下線が表示されます。

別名 U

ODS 出力先 HTML、RTF、PRINTER

ヒント ODS では値 1、2、3 に対して同じ種類の下線が生成されます。SAS/GRAPH では、値 1、2、3 に従って生成される下線の幅が徐々に太くなります。

注 デフォルトでは、ODS での TITLE ステートメントの表示設定は、現在のスタイルのシステムタイトルに関連するスタイル要素から取得します。TITLE ステートメントの構文で *ods-format-options* を指定すると、現在のスタイルから提供される設定より優先されます。現在のスタイルは、ODS 出力先によって異なります。

ヒント これらのオプションを文字、1 単語、複数単語ごとに指定することができます。オプションに続けて、*text* にそれぞれの文字や単語を指定します。

たとえば、次のコードでは、タイトルテキスト“Red, White, and Blue”が異なる色で表示されます。

```
title color=red "Red," color=white "White, and" color=blue "Blue";
```

### 'text' | "text"

一重引用符または二重引用符で囲んだテキストを指定します。

PROC ステップに指定したタイトルに BY 変数の値(#BYVALn)、BY 変数名(#BYVARn)、BY 行(#BYLINE)を挿入すると、タイトルをカスタマイズできます。指定したタイトルテキスト文字列内で、代替テキストを表示する位置にこれらの項目を埋め込みます。

**#BYVAL $n$  | #BYVAL(variable-name)**

テキスト文字列に指定した#BYVAL を BY 変数の現在の値に置き換えて、タイトルに表示します。

PROC ステップの TITLE ステートメントで#BYVAL を使用する場合は、次のルールに従います。

- BY ステートメントの#BYVAL で使用した変数を指定します。
- 指定したタイトルテキスト文字列内で、置き換えたテキストを表示する位置に#BYVAL を挿入します。
- #BYVAL の後ろに、テキスト文字列の区切りを示す、空白か他の英数字以外の文字(例:引用符)のいずれかの区切り文字を指定します。
- #BYVAL の置き換えのすぐ後ろに他のテキストを追加する場合は、区切り文字ではなく、マクロ変数と同じように末尾にドットを使用します。

次のいずれかの値の変数を指定します。

*n*

#BYVAL に BY ステートメントのどの変数を使用するかを指定します。*n* の値は、BY ステートメントの変数の位置を示します。

例 **#BYVAL2** では、BY ステートメントの 2 番目の変数を指定します。

**variable-name**

BY 変数の名前を指定します。

ヒント *variable-name* では、大文字と小文字は区別されません。

例 **#BYVAL(YEAR)** では、BY 変数 YEAR を指定します。

**#BYVAR $n$  | #BYVAR(variable-name)**

テキスト文字列に指定した#BYVAR を BY 変数の名前または変数に関連付けられているラベルで置き換え、タイトルに名前またはラベルを表示します。

PROC ステップの TITLE ステートメントで#BYVAR を使用する場合は、次のルールに従います。

- BY ステートメントの#BYVAR で使用した変数を指定します。
- 指定したタイトルテキスト文字列内で、代替テキストを表示する位置に#BYVAR を挿入します。
- #BYVAR の後ろに、テキスト文字列の区切りを示す、空白か他の英数字以外の文字(例:引用符)のいずれかの区切り文字を指定します。
- #BYVAR の置き換えに続けて他のテキストを追加する場合、区切り文字ではなく、マクロ変数と同じようにドットを末尾に使用します。

次のいずれかの値の変数を指定します。

*n*

#BYVAR に BY ステートメントのどの変数を使用するかを指定します。*n* の値は、BY ステートメントの変数の位置を示します。

例 **#BYVAR2** では、BY ステートメントの 2 番目の変数を指定します。

**variable-name**

BY 変数の名前を指定します。



ヒント `variable-name` では、大文字と小文字は区別されません。

例 **#BYVAR(SITES)**では、BY 変数 SITES を指定します。

### #BYLINE

テキスト文字列に指定した#BYLINE を先頭と末尾に空白を含まない BY 行全体で置き換えて、タイトルに表示します。

ヒント #BYLINE を指定すると、OPTIONS ステートメントに NOBYLINE を指定して出力を抑制しない限り、ページ上部に BY 行を含む出力が生成されます。

参照項目: NOBYLINE の詳細については、“[BYLINE System Option](#)” (SAS [Viya System Options: Reference](#))を参照してください。

ヒント 以前のリリースとの互換性を保つため、引用符で囲まれていない一部のテキストが受け入れられます。新しいプログラムを作成したり、既存のプログラムを更新する場合は、指定するテキストを常に引用符で囲むようにしてください。

マクロやマクロ変数を使用すると、TITLE ステートメント内の情報を変更できます。タイトルが二重引用符(“”)で囲まれている場合、そのテキストはタイトルに置き換えられます。タイトルが一重引用符(”)で囲まれている場合、テキストは置き換えられません。

マクロやマクロ変数を使用すると、TITLE ステートメント内の情報を変更できます。SAS マクロ機能が、このマクロ変数を解決します。

## 詳細

TITLE ステートメントはステップまたはステップに関連付けられた RUN グループの実行時に有効になります。ある行に対して指定したタイトルは、指定したタイトルを取り消すか、その行に別のタイトルを定義するまで、後続のすべての出力でそのタイトルが使用されます。特定の行に対して TITLE ステートメントを実行すると、その行に設定されていた以前の TITLE ステートメントや、行番号が  $n$  番目以降のすべての行が取り消されます。

### 動作環境の情報

対話モードの場合、タイトルの最大長は 254 文字です。その他の場合、最大長は 200 文字です。指定されたタイトルの長さが LINESIZE オプションの値よりも大きい場合、そのタイトルは行のサイズに収まるように切り捨てられます。

## 例

### 例 1: TITLE ステートメントの使用

次の例では TITLE ステートメントを使用する方法を示します。

- 次のステートメントは、行番号  $n$  とそれ以降のすべての行でタイトルの出力を抑制します。

```
titlen;
```

- 次のコード行では、TITLE ステートメントの例を示します。

```
title 'First Draft';
```



- title2 "Year's End Report";
- title2 'Year"s End Report';

### 例 2: BY 変数値を使用したタイトルのカスタマイズ

PROC ステップに指定したタイトルに BY 変数の値を挿入して、タイトルをカスタマイズすることができます。次の例では、#BYVALn、#BYVARn、および #BYLINE の使用方法を示します。

- title 'Quarterly Sales for #byval(site);
- title 'Annual Costs for #byvar2';
- title 'Data Group #byline';

### 例 3: ODS (Output Delivery System)を使用したタイトルとフットノートのカスタマイズ

ODS を用いてタイトルとフットノートをカスタマイズできます。次の例では、PROC TEMPLATE を使用してタイトルとフットノートの色、位置調整、テキストサイズを変更する方法を示します。

```

/*****
 *The following program creates the data set *
 *grain_production and the $cntry format. *
 *****/
data grain_production;
  length Country $ 3 Type $ 5;
  input Year country $ type $ Kilotons;
  datalines;
1995 BRZ Wheat 1516
1995 BRZ Rice 11236
1995 BRZ Corn 36276
1995 CHN Wheat 102207
1995 CHN Rice 185226
1995 CHN Corn 112331
1995 IND Wheat 63007
1995 IND Rice 122372
1995 IND Corn 9800
1995 INS Wheat .
1995 INS Rice 49860
1995 INS Corn 8223
1995 USA Wheat 59494
1995 USA Rice 7888
1995 USA Corn 187300
1996 BRZ Wheat 3302
1996 BRZ Rice 10035
1996 BRZ Corn 31975
1996 CHN Wheat 109000
1996 CHN Rice 190100
1996 CHN Corn 119350
1996 IND Wheat 62620
1996 IND Rice 120012
1996 IND Corn 8660
1996 INS Wheat .
1996 INS Rice 51165
1996 INS Corn 8925
1996 USA Wheat 62099

```

```

1996 USA Rice 7771
1996 USA Corn 236064
;
run;
proc format;
  value $cntry 'BRZ'='Brazil'
              'CHN'='China'
              'IND'='India'
              'INS'='Indonesia'
              'USA'='United States';
run;

/*****
*This PROC TEMPLATE step creates the *
*table definition TABLE1 that is used *
*in the DATA step. *
*****/
proc template;
  define table table1;
    mvar sysdate9;
    dynamic colhd;
    classlevels=on;
  define column char_var;
    generic=on;
    blank_dups=on;
    header=colhd;
    style=cellcontents;
  end;

  define column num_var;
    generic=on;
    header=colhd;
    style=cellcontents;
  end;

  define footer table_footer;
  end;
end;
run;

/*****
*The ODS HTML statement creates HTML output created with *
*the style definition D3D. *
* *
*The TITLE statement specifies the text for the first title *
*and the attributes that ODS uses to modify it. *
*The J= style attribute left-justifies the title. *
*The COLOR= style attributes change the color of the title text *
*"Leading Grain" to blue and "Producers in" to green. *
* *
*The TITLE2 statement specifies the text for the second title *
*and the attributes that ODS uses to modify it. *
*The J= style attribute center justifies the title. *
*The COLOR= attribute changes the color of the title text "2010" *
*to red. *
* The HEIGHT= attributes change the size of each *
*individual number in "2010". *
*****/

```

```

*
*
*The FOOTNOTE statement specifies the text for the first footnote *
*and the attributes that ODS uses to modify it. *
*The J=left style attribute left-justifies the footnote. *
*The HEIGHT=20 style attribute changes the font size to 20pt. *
*The COLOR= style attributes change the color of the footnote text *
*"Prepared" to red and "on" to green. *
*
*
*The FOOTNOTE2 statement specifies the text for the second footnote *
*and the attributes that ODS uses to modify it. *
*The J= style attribute centers the footnote. *
*The COLOR= attribute changes the color of the date *
*to blue, *
*The HEIGHT= attribute changes the font size *
*of the date specified by the sysdate9 macro. *
*****/
ods html body='newstyle-body.htm'
style=d3d;

title j=left
font= 'Times New Roman' color=blue bcolor=red "Leading Grain "
c=green bold italic "Producers in";
title2 j=center color=red underlin=1
height=28pt "1"
height=24pt "9"
height=20pt "9"
height=16pt "6";

footnote j=left height=20pt
color=red "Prepared "
c='#FF9900' "on";
footnote2 j=center color=blue
height=24pt "&sysdate9";
footnote3 link='http://support.sas.com' "SAS";
/*****
*This step uses the DATA step and ODS to produce *
*an HTML report. It uses the default table definition *
*(template) for the DATA step and writes an output object *
*to the HTML destination. *
*****/
data _null_;
set grain_production;
where type in ('Rice', 'Corn') and year=1996;
file print ods=(
template='table1'
columns=(
char_var=country(generic=on format=$cntry.
dynamic=(colhd='Country'))
char_var=type(generic dynamic=(colhd='Year'))
num_var=kilotons(generic=on format=comma12.
dynamic=(colhd='Kilotons'))
)
);

put _ods_;
run;

```

アウトプット 2.31 カスタマイズしたタイトルとフットノートを使用した出力

**関連項目:****ステートメント:**

- “FOOTNOTE ステートメント” (125 ページ)

**システムオプション:**

- “LINESIZE= System Option” (*SAS Viya System Options: Reference*)

---

**UPDATE ステートメント**

トランザクションを適用してマスタファイルを更新します。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	実行
制限事項:	このステートメントは CAS Server ではサポートされていません。
注:	UPDATE ステートメントを使用して読み込まれた値は PDV で保持されます。読み込まれた変数のデータタイプも保持されます。詳細については、“ <a href="#">RETAIN ステートメント</a> ” (361 ページ)を参照してください。
注意:	UPDATE ステートメントを使用する際に OUTPUT ステートメントを追加すると、望ましくない結果が生成されます。

## 構文

```
UPDATE master-data-set <(data-set-options)> transaction-data-set <(data-set-options)>
    <END=variable>
    <UPDATERMODE=MISSINGCHECK | NOMISSINGCHECK>;
BY by-variable;
```

## 引数

### *master-data-set*

マスタファイルとして使用する SAS データセットを指定します。

**範囲** この名前は 1 レベル名(例:FITNESS)、2 レベル名(例:IN.FITNESS)、および特殊 SAS データセット名のいずれでもかまいません。

**ヒント** データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

### *(data-set-options)*

DATA ステップに処理する変数を読み込むときに実行するアクションを指定します。

**要件** *Data-set-options* は SAS データセット名の後に丸かっこで囲んで指定する必要があります。

**ヒント** データセットを更新する場合、変数の除外、変数の指定、変数名の変更を実行すると便利な場合があります。同じ名前の変数名を変更すると、同じ名前を持つ最初の変数の値が、同じ名前を持つ 2 番目の変数の値で上書きされるのを回避することができます。このように変数名の 1 つを変更すると、比較などの処理で両方の変数の値を使用できるようになります。

**参照項目:** 入力データセットで使用するデータセットオプションのリストについては、次のドキュメントを参照してください。 *SAS Viya Data Set Options: Reference*

**例** “[例 2: 変数名を変更する更新](#)” (412 ページ)

### *transaction-data-set*

マスタデータセットに適用する変更を含む SAS データセットを指定します。

**範囲** この名前は 1 レベル名(例:HEALTH)、2 レベル名(例:IN.HEALTH)、および特殊 SAS データセット名のいずれでもかまいません。

**ヒント** データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パ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。

### **END=variable**

作成する一時変数の名前を指定します。この変数の値には終端指示子が格納されます。この変数の値は 0 に初期化され、UPDATE ステートメントで最後のオブザベーションを処理するときに 1 に設定されます。この変数はどのデータセットにも追加されません。

### **UPDATEMODE=MISSINGCHECK**

### **UPDATEMODE=NOMISSINGCHECK**

トランザクションデータセットにある変数の欠損値を使用して、マスタデータセットにある既存の変数の値を置き換えるかどうかを指定します。

### **MISSINGCHECK**

トランザクションデータセットにある変数の欠損値を使用して、マスタデータセットにある変数の値を置き換えないように指示します。

### **NOMISSINGCHECK**

トランザクションデータセットにある変数の欠損値を使用して、マスタデータセットにある変数の値を置き換えるように指示します。

**デフォルト** MISSINGCHECK

**ヒント** ただし、特殊欠損値は例外です。この場合、MISSINGCHECK(デフォルト)が有効な場合でも、マスタデータセットの値は特殊欠損値で置き換えられます。

## **詳細**

### **要件**

- UPDATE ステートメントは、必ず BY ステートメントとともに指定します。BY ステートメントには、オブザベーションを識別するための変数を指定します。
- BY ステートメントは適用する UPDATE ステートメントの直後に指定する必要があります。
- UPDATE ステートメントにリストされているデータセットは、BY ステートメントにリストされている変数の値でソートされているか、適切なインデックスを含んでいる必要があります。
- マスタデータセットの各オブザベーションには、重複しない BY 変数の値が含まれている必要があります。複数の BY 変数の値がある場合、その値の最初のオブザベーションのみが更新されます。トランザクションデータセットには、同一の BY 変数の値を持つ複数のオブザベーションが含まれていてもかまいません。(複数のトランザクションオブザベーションは、出力ファイルに書き込まれる前に、マスタオブザベーションにすべて適用されます。)

## トランザクションデータセット

通常、マスタデータセットとトランザクションデータセットには同じ変数が含まれています。ただし、処理時間を短縮するために、更新対象の変数のみを含むトランザクションデータセットを作成することができます。また、トランザクションデータセットには出力データセットに追加する新しい変数を含めることもできます。

出力データセットには、マスタデータセット内のオブザベーションごとに 1 つのオブザベーションが含まれます。トランザクションデータセットにマスタデータセットと一致しないオブザベーションが存在する場合、一致しないオブザベーションは新しいオブザベーションとして出力データセットに書き込まれます。更新対象ではないオブザベーションはトランザクションデータセットから省くことができます。

## 欠損値

デフォルトでは、UPDATEMODE=MISSINGCHECK オプションが有効です。そのため、トランザクションデータセットの欠損値によって、マスタデータセットの既存の値は置き換えられません。そのため、すべての変数ではなく一部の変数のみを更新する場合、またオブザベーションごとに更新する変数が異なる場合は、変更しない変数を欠損値に設定します。トランザクションデータに含まれる欠損値でマスタデータセットに含まれる既存の値を置き換える場合は、UPDATEMODE=NOMISSINGCHECK を使用します。

UPDATEMODE=MISSINGCHECK が有効な場合でも、トランザクションデータセットで特殊欠損値の文字を使用すると、既存の値を欠損値に置き換えることができます。特殊欠損値を含むトランザクションデータセットを作成するには、DATA ステップで MISSING ステートメントを使用します。トランザクションデータセットに **A** から **Z** までの特殊欠損値の 1 つを定義すると、マスタデータセットに含まれる数値変数はその値に更新されます。

マスタデータセットの値を通常欠損値に設定する場合は、トランザクションデータセット内でアンダースコア(\_)を 1 つ使用して欠損値を示します。このように指定すると、マスタデータセットの値は、数値の欠損値がピリオド(.)に設定され、文字の欠損値はブランクに設定されます。

特殊欠損値の定義方法および使用方法の詳細については、“[MISSING ステートメント](#)” (288 ページ)を参照してください。

## 比較

- UPDATE ステートメントと MERGE ステートメントは、どちらも SAS データセットのオブザベーションを更新します。
- MERGE ステートメントは、最初のデータセットにある既存の値を 2 番目のデータセットにある欠損値に自動的に置き換えます。ただし、UPDATE ステートメントは、デフォルトではこのように動作しません。UPDATE ステートメントを使用してマスタデータセットの既存の値をトランザクションデータセットの欠損値で上書きするには、UPDATEMODE=NOMISSINGCHECK を使用する必要があります。
- UPDATE ステートメントでは、トランザクションを適用することによって、マスタファイル内の選択したオブザベーションの値のみを変更したり、更新することができます。UPDATE ステートメントでは、新しいオブザベーションを追加することもできます。

## 例

### 例 1: 基本的な更新

次のプログラムステートメントでは、マスタデータセット(OHIO.JAN)にトランザクションを適用して、新しいデータセット(OHIO.QTR1)を作成します。BY 変数 STORE は、OHIO.JAN と OHIO.WEEK4 の両方に存在し、マスタデータセットでその値は重複しない必要があります。

```
data ohio.qtr1;
  update ohio.jan ohio.week4;
  by store;
run;
```

### 例 2: 変数名を変更する更新

この例は、データセットの変数名を変更したため、プログラムデータベクトル内の同じ変数が上書きされないことを示しています。また、各データセットで WEIGHT 変数の名前が変更され、新しい WEIGHT 変数の値が算出されます。マスタデータセットとトランザクションデータセットは更新を実行するコードの前にリストされています。

```
Master Data Set
  HEALTH
OBS ID NAME TEAM WEIGHT
1 1114 sally blue 125
2 1441 sue green 145
3 1750 joey red 189
4 1994 mark yellow 165
5 2304 joe red 170

Transaction Data Set
  FITNESS
OBS ID NAME TEAM WEIGHT
1 1114 sally blue 119
2 1994 mark yellow 174
3 2304 joe red 170
/*****/

data health;
  input ID NAME $ TEAM $ WEIGHT;
  length team $ 6;
  cards;
1114 sally blue 125
1441 sue green 145
1750 joey red 189
1994 mark yellow 165
2304 joe red 170
;
data fitness;
  input ID NAME $ TEAM $ WEIGHT;
  length team $ 6;
  cards;
1114 sally blue 119
1994 mark yellow 174
2304 joe red 170
;
```



```

/* Sort both data sets by ID */
proc sort data=health;
  by id;
run;
proc sort data=fitness;
  by id;
run;
/* Update Master with Transaction */
data health2;
  length STATUS $11;
  update health(rename=(weight=ORIG) in=a)
    fitness(drop=name team in=b);
  by id ;
  if a and b then
  do;
    CHANGE=abs(orig - weight);
    if weight<orig then status='loss';
    else if weight>orig then status='gain';
    else status='same';
  end;
  else status='no weigh in';
run;

proc print data=health2;
  title 'Weekly Weigh-in Report';
run;

```

#### アウトプット 2.32 変数名を変更する更新

Obs	STATUS	ID	NAME	TEAM	ORIG	WEIGHT	CHANGE
1	loss	1114	sally	blue	125	119	6
2	no weigh in	1441	sue	green	145	.	.
3	no weigh in	1750	joey	red	189	.	.
4	gain	1994	mark	yellow	165	174	9
5	same	2304	joe	red	170	170	0

#### 例 3: 欠損値による更新

この例は、マスタデータセット PAYROLL と、通常の欠損値と特殊欠損値を含むデータセット INCREASE を作成する DATA ステップを示しています。更新後の値は次のようになります。

- ID 1026 の給与の値は変更されません。
- ID 1034 の給与の値は特殊欠損値になります。
- ID 1057 の給与の値は通常の欠損値になります。

```

/* Create the Master Data Set */
data payroll;
  input ID SALARY;

```

```

    datalines;
1011 245
1026 269
1028 374
1034 333
1057 582
;
/* Create the Transaction Data Set */
data increase;
input ID SALARY;
missing A _;
datalines;
1011 376
1026 .
1028 374
1034 A
1057 _
;
/* Update Master with Transaction */
data newpay;
update payroll increase;
by id;
run;
proc print data=newpay;
title 'Updating with Missing Values';
run;

```

アウトプット 2.33 欠損値による更新

**Updating with Missing Values**

Obs	ID	SALARY
1	1011	376
2	1026	269
3	1028	374
4	1034	A
5	1057	.

### 関連項目:

- [“Definition of Data Set Options” \(SAS Viya Data Set Options: Reference\)](#)

### ステートメント:

- [“BY ステートメント” \(33 ページ\)](#)
- [“MERGE ステートメント” \(280 ページ\)](#)
- [“MISSING ステートメント” \(288 ページ\)](#)
- [“MODIFY ステートメント” \(289 ページ\)](#)
- [“SET ステートメント” \(378 ページ\)](#)

**システムオプション:**

- [“MISSING= System Option” \(SAS Viya System Options: Reference\)](#)

## WAITFOR ステートメント

タスクの実行が終了するまで、現在の SAS セッションの実行を中断します。

**該当要素:** 任意の場所

**カテゴリ:** 動作環境

**制限事項:** このステートメントは CAS Server ではサポートされていません。

### 構文

```
WAITFOR <_ANY_ | _ALL_> taskname <taskname...> <TIMEOUT=seconds>;
```

### 必須引数

***taskname***

ユーザーが待機するタスクの名前を指定します。タスク名については、[“SYSTASK ステートメント” \(396 ページ\)](#)を参照してください。指定したタスク名は、SYSTASK COMMAND ステートメントを通じて割り当てられたタスク名と完全に一致している必要があります。タスク名を指定する際にワイルドカードは使用できません。

### オプション引数

**\_ANY\_ | \_ALL\_**

現在の SAS セッションの実行を、指定された 1 つまたは全部のタスクの実行が終了するまで中断します。デフォルトの設定は \_ANY\_ です。これは、指定されたタスクのうちの 1 つが実行を完了するやいなや、WAITFOR ステートメントが実行を終了することを意味しています。

**TIMEOUT=*seconds***

WAITFOR が現在の SAS セッションを中断する最大秒数を指定します。TIMEOUT オプションを指定しない場合、WAITFOR は SAS セッションの実行を無期限に中断します。

### 詳細

WAITFOR ステートメントは、指定されたタスクの実行が終了するか TIMEOUT= の間隔時間が経過するまで、現在の SAS セッションを中断します。指定されたタスクが WAIT オプションで開始された場合、WAITFOR ステートメントはそのタスクを無視します。WAIT オプションについては、[“SYSTASK ステートメント” \(396 ページ\)](#)を参照してください。

たとえば次のステートメントは、3 つの異なる X クライアントプログラムを開始し、それらが完了するのを待機します。

```
systask command "xv" taskname=pgm1;
systask command "xterm" taskname=pgm2;
systask command "xcalc" taskname=pgm3;
waitfor _all_ pgm1 pgm2 pgm3;
```

WAITFOR ステートメントを使用して、同時に複数の SAS セッションを実行できます。次のステートメントは 3 つの異なる SAS ジョブを開始し、それら 3 つのジョブの実行が終了するまで現在の SAS セッションの実行を中断します。

```
systask command "sas myprog1.sas" taskname=sas1;
systask command "sas myprog2.sas" taskname=sas2;
systask command "sas myprog3.sas" taskname=sas3;
waitfor _all_ sas1 sas2 sas3;
```

注: この方法では、SAS は各コマンドの後に終了するため、パフォーマンスが低下します。SAS/CONNECT を使用して、並行する SAS セッションを実行することもできます。詳細については、*SAS/CONNECT for SAS Viya: User's Guide* を参照してください。

SYSTASK コマンドを複数回使用する長期のジョブがある場合は、SYSTASK コマンドで WAITFOR ステートメントまたは CLEANUP オプションを使用してメモリをクリアします。WAITFOR ステートメントは、SYSTASK コマンドによって開始されたすべての完了プロセスの情報を削除することによってメモリを解放します。CLEANUP オプションは、特定のジョブが完了したときにメモリをクリアし、今後の使用に備えてメモリを開放します。ジョブが完了した後に WAITFOR ステートメントを使用する場合、そのジョブはすでに CLEANUP オプションによってクリーンアップされているので、このステートメントは無効になります。

SAS マクロ変数 SYSRC には、WAITFOR ステートメントのリターンコードが含まれます。WAITFOR ステートメントの実行が失敗する場合、SYSRC マクロ変数に非ゼロ値が含まれていることがあります。たとえば、WAITFOR ステートメントに構文エラーが含まれている場合があります。TIMEOUT オプションで指定された秒数が経過した場合は WAITFOR ステートメントの実行が終了し、次のいずれかのアクションが発生した場合は SYSRC が非ゼロ値に設定されます。

- 実行が終了しないタスクを 1 つ指定します。
- 複数のタスクと `_ANY_` オプション(デフォルトの設定)を指定しますが、それらのタスクの実行はいずれも終了しません。
- 複数のタスクと `_ALL_` オプションを指定しますが、それらのタスクのうち 1 つの実行が終了しません。

WAITFOR ステートメントが実行された後もステータス変数が依然として NULL であるタスクは、実行が完了していません。個別のタスクのステータス変数に関する説明は、“[SYSTASK ステートメント](#)” (396 ページ)を参照してください。

### 関連項目:

- [SAS/CONNECT for SAS Viya: User's Guide](#)
- [“Executing Operating System Commands from Your SAS Session” \(Batch and Line Mode Processing in SAS Viya\)](#)

### ステートメント:

- [“SYSTASK ステートメント” \(396 ページ\)](#)
- [“X ステートメント” \(423 ページ\)](#)

---

## WHERE ステートメント

SAS データセットから特定の条件を満たすオブザベーションを選択します。

**該当要素:** DATA ステップまたは PROC ステップ

**カテゴリ:** アクション

**種類:** 宣言

**制限事項:** このステートメントは CAS Server ではサポートされていません。

**ヒント:** WHERE ステートメントを使用してデータをフィルタリングする方法については、SAS チュートリアルビデオデータのフィルタリングをご覧ください。

---

## 構文

**WHERE** *where-expression-1*  
< *logical-operator where-expression-n* >;

## 引数

### *where-expression*

オペランドや演算子で構成される算術式または論理式を指定します。

**ヒント** 次のいくつかのセクションで説明されるオペランドや演算子は WHERE=データセットオプションでも使用できます。

---

*where-expression* を複数指定することができます。

---

### *logical-operator*

AND、AND NOT、OR、OR NOT を指定できます。

## 詳細

### 基本

WHERE ステートメントを使用すると、入力データセットからすべてのオブザベーションを読み込む必要がなくなるため、SAS プログラムの処理効率が向上します。

WHERE ステートメントは条件を指定して実行することはできません。そのため、IF-THEN ステートメントの中で使用することはできません。

WHERE ステートメントには、論理演算子で連結した複数の WHERE 式を指定できます。

**注:** WHERE 式を使用して SAS データセットにあるオブザベーションのサブセットにアクセスするときに、インデックス付きの SAS データセットを使用するとパフォーマンスが大幅に向上します。

### DATA ステップでの使用

WHERE ステートメントは、直前にある SET、MERGE、MODIFY、UPDATE の各ステートメントに指定されているすべてのデータセットに適用されます。

WHERE ステートメントで使用する変数は、WHERE ステートメントが適用されるすべてのデータセットに含まれている必要があります。WHERE ステートメントは、POINT=オプションを指定した SET ステートメントや MODIFY ステートメントとともに使用することはできません。

OBS=および FIRSTOBS=の処理を WHERE ステートメントの処理に適用できません。

WHERE ステートメントを使用して、生データを含む外部ファイルからレコードを選択することはできません。また、DATA LINES ステートメントでインストリ

ームデータを読み込む DATA ステップ内に WHERE ステートメントは使用できません。

DATA ステップを繰り返すたびに、SET、MERGE、MODIFY、UPDATE の各ステートメントの最初の操作として、入力データセット内のオブザベーションが WHERE ステートメントの条件を満たしているかどうかの確認が実行されます。入力データセットオプションの適用直後、DATA ステップ内の他のステートメントを実行する前に、WHERE ステートメントが適用されます。DATA ステップでのオブザベーションの結合に MERGE、MODIFY、UPDATE ステートメントとともに WHERE ステートメントを使用する場合、各入力データセットからオブザベーションを選択した後で結合します。

### DATA ステップ内の WHERE ステートメントと BY ステートメント

DATA ステップに WHERE ステートメントと BY ステートメントの両方が含まれている場合、BY グループを作成する前に WHERE ステートメントが実行されます。そのため、BY グループは、WHERE ステートメントで選択したオブザベーションサブセットのオブザベーショングループを反映します。元の入力データセットのオブザベーションの現行の BY グループを反映しません。

### PROC ステップ

WHERE ステートメントは、SAS データセットを読み込む SAS プロシジャとともに使用できます。WHERE ステートメントは、プロシジャで処理する元のデータセットをサブセット化する場合に便利です。SAS *Viya Data Management and Utility Procedures Guide* では、複数のデータセットの指定可能なプロシジャでの WHERE ステートメントのアクションのみが説明されています。他の場合、WHERE ステートメントはこのトピックで説明されているように動作します。

### インデックスの使用

次の演算子と関数のいずれかと組み合わせてインデックス付き変数が使用されている場合、DATA ステップまたは PROC ステップは利用可能なインデックスを使用してデータの選択を最適化しようとします。

- BETWEEN-AND 演算子
- 比較演算子(コロン修飾子あり/なし)
- CONTAINS 演算子
- IS NULL および IS NOT NULL 演算子
- LIKE 演算子
- TRIM 関数
- SUBSTR 関数(場合による)

SUBSTR 関数の引数が次のすべての条件を満たす場合、インデックスが処理に使用されます。

```
where substr(variable,position,length)
='character-string';
```

SUBSTR 関数の引数が次のすべての条件を満たす場合、インデックスが処理に使用されます。

- *position* が 1 に等しい
- *length* が *variable* の長さよりも短い、またはその長さに等しい
- *length* が *character-string* の長さに等しい

## WHERE 式で使用されるオペランド

WHERE 式で使用されるオペランドには次の値が含まれます。

- 定数
- 日付と時刻の値
- SAS データセットから取得した変数の値
- WHERE 式自身で作成された値

DATA ステップで作成される変数は WHERE 式で使用できません(たとえば、FIRST.variable、LAST.variable、\_N\_、または割り当てステートメントで作成される変数)。これは、DATA ステップまたは PROC ステップにオブザベーションを読み込む前に WHERE ステートメントが実行されるためです。WHERE 式に比較演算子を使用されている場合、フォーマットされていない変数の値が比較されません。

WHERE 式でオペランドを使用する例を次に示します。

- where score>50;
- where date>='01jan1999'd and time>='9:00't;
- where state='Mississippi';

他の SAS 式の場合と同様に、数値変数の名前は単独で指定できます。この場合、値が 0 または欠損値の場合は偽になります。その他の値の場合は真になります。これらの例では、WHERE 式に数値変数 EMPNUM と SSN が含まれています。

- where empnum;
- where empnum and ssn;

WHERE 式では、文字リテラルまたは文字変数の名前も単独で使用できます。WHERE 式として文字変数を単独で指定した場合、文字変数の値が空ではないオブザベーションが選択されます。

## WHERE 式で使用される演算子

WHERE ステートメントには SAS 演算子と WHERE 式の特殊な演算子を使用できます。演算子の詳細なリストについては、[表 2.12 \(419 ページ\)](#)を参照してください。

WHERE ステートメントの演算子を次の表に示します。

表 2.12 WHERE ステートメント演算子

演算子の種類	記号またはニーモニック	説明
算術演算子		
	*	乗算
	/	除算
	+	加算
	-	減算
	**	累乗

演算子の種類	記号またはニーモニック	説明
比較演算子†		
	= または EQ	等しい
	^=、≠、≠、または NE*	等しくない
	> または GT	より大きい
	< または LT	未満
	>= または GE	以上
	<= または LE	以下
	IN	リストのどれかと等しい
論理演算子(ブール演算子)		
	& または AND	論理積
	または OR**	論理和
	~, ^、≠、または NOT*	論理否定
その他の演算子		
	***	文字変数の連結
	()	評価の順序を示す
	+ 接頭辞	正の数
	- 接頭辞	負の数
WHERE 式のみ		
	BETWEEN-AND	対象範囲
	? または CONTAINS	文字列
	IS NULL または IS MISSING	欠損値
	LIKE	パターンマッチ
	=*	類似



演算子の種類	記号またはニーモニック	説明
	SAME-AND	元の句を再入力せずに、既存の WHERE ステートメントに句を追加

- \* キャレット(^)、チルダ(~)、否定(!)はすべて論理否定を示しています。お使いのキーボード上の文字を使用するか、または対応するニーモニックを使用します。
- \*\* OR 記号(|)、ブロークンバーティカルバー(| |)、感嘆符(!)はすべて論理和を表します。お使いのキーボード上の文字を使用するか、または対応するニーモニックを使用します。
- \*\* 連続する2つの OR 記号(| |)、ブロークンバーティカルバー(| |)、感嘆符(!!)はすべて連結を表します。お使いのキーボード上の文字を使用します。
- † 比較演算子とともにコロン修飾子(:)を使用すると、文字列にある指定した接頭辞のみを比較できます。

## 比較

- SET、MERGE、MODIFY、UPDATE の各ステートメントで複数のデータセットを指定するときに個々のデータセットからオブザベーションを選択するには、WHERE=データセットオプションを各データセットに適用する必要があります。DATA ステップ内で WHERE ステートメントと WHERE=データセットオプションが同じデータセットに適用される場合、WHERE=データセットオプションが優先され、WHERE ステートメントは無視されます。WHERE データセットオプションが適用されていない他のデータセットでは、WHERE ステートメントが使用されます。
- DATA ステップにおける WHERE ステートメントとサブセット化 IF ステートメントの重要な違いを次に示します。
  - WHERE ステートメントでは、オブザベーションがプログラムデータベクトルに読み込まれる前にオブザベーションを選択します。そのため、プログラムの処理効率が向上します。サブセット化 IF ステートメントでは、オブザベーションがプログラムデータベクトルに読み込まれた後に、オブザベーションを操作します。
  - BY ステートメントが SET、MERGE、UPDATE の各ステートメントとともに使用される場合、WHERE ステートメントとサブセット化 IF ステートメントは異なるデータセットを作成します。データセットに違いが発生するのは、サブセット化 IF ステートメントではオブザベーションを選択する前に BY グループが作成され、WHERE ステートメントではオブザベーションを選択した後に BY グループが作成されるためです。
  - WHERE ステートメントは IF ステートメントの一部として条件を指定して実行できませんが、サブセット化 IF ステートメントは条件を指定して実行できます。
  - WHERE ステートメントでは、SAS データセットにあるオブザベーションのみを選択します。ただし、サブセット化 IF ステートメントでは、既存のデータセットや INPUT ステートメントで作成されたオブザベーションからもオブザベーションを選択します。
  - SAS ウィンドウ環境プロシジャでは、サブセット化 IF ステートメントを使用してオブザベーションをサブセット化して、参照および参照できません。
- WHERE ステートメントを DROP または KEEP ステートメントと混同しないでください。DROP と KEEP ステートメントでは、処理対象の変数を選択します。WHERE ステートメントでは、オブザベーションを選択します。

## 例

### 例 1: WHERE ステートメントの基本的な使用法

次の DATA ステップは、データセット CUSTOMER から NAME が **Mac** で始まり、CITY の値が **Charleston** または **Atlanta** であるオブザベーションのみを含む SAS データセットを作成します。

```
data testmacs;
  set customer;
  where substr(name,1,3)='Mac' and
    (city='Charleston' or city='Atlanta');
run;
```

### 例 2: WHERE ステートメントのみで使用可能な演算子を使用する

- BETWEEN-AND 演算子の使用:

```
where empnum between 500 and 1000;
```

- CONTAINS 演算子の例:

```
where company ? 'bay';
where company contains 'bay';
```

- IS NULL および IS MISSING 演算子の例:

```
where name is null;
where name is missing;
```

- 文字 D で始まるすべての名前を選択する LIKE 演算子の例:

```
where name like 'D%';
```

- 次の名前リストからパターンマッチを行う LIKE 演算子の例:

```
Diana
Diane
Dianna
Dianthus
Dyan
```

WHERE ステートメント	選択される名前
where name like 'D_an';	Dyan
where name like 'D_an_';	Diana、Diane
where name like 'D_an__';	Dianna
where name like 'D_an%';	リストに含まれるすべての名前

- SOUNDS-LIKE 演算子を使用すると、“Smith”に似た名前が選択されます。

```
where lastname=*'Smith';
```

- SAME-AND の使用:

```
where year>1991;
...more SAS statements...
```

```
where same and year<1999;
```

この例にある 2 番目の WHERE ステートメントは、次の WHERE ステートメントに相当します。

```
where year>1991 and year<1999;
```

## 関連項目:

### データセットオプション:

- [“WHERE= Data Set Option” \(SAS Viya Data Set Options: Reference\)](#)

### ステートメント:

- [“IF ステートメント、サブセット化” \(134 ページ\)](#)

---

## X ステートメント

SAS セッションから動作環境のコマンドを発行します。

**該当要素:** 任意の場所

**カテゴリ:** 動作環境

**制限事項:** このステートメントは CAS Server ではサポートされていません。

## 構文

```
X <'operating-environment-command'>;
```

### 引数なし

引数を指定せずに X ステートメントを使用すると、ユーザーはその動作環境に置かれるため、動作環境固有のコマンドを発行できます。

### 引数

**'operating-environment-command'**

動作環境のコマンドを引用符で囲んで指定します。

#### 動作環境の情報

Linux コマンドを指定します。Linux コマンドを 1 つのみを指定する場合は、引用符で囲む必要はありません。また、Korn シェルから SAS を実行している場合は、別名は使用できません。

## 詳細

### 概要情報

ウィンドウ環境または対話型ラインモードで SAS を実行中のときは、X ステートメントをすべての動作環境で使用できます。一部の動作環境では、バッチモードまたは非対話型モードで SAS を実行している場合でも X ステートメントを使用できることがあります。動作環境モードから SAS セッションに戻る方法は、動作環境や X ステートメントで使用した動作環境固有のコマンドによって異なります。

X ステートメントを SAS マクロとともに使用すると、複数の動作環境で実行可能な SAS プログラムを作成できます。詳細については、*SAS Viya マクロ言語リファレンス*を参照してください。

### Linux 固有の情報

X ステートメントは、SAS セッションの内部から Linux コマンドを発行します。SAS は X ステートメントを即座に実行します。

X ステートメントも %SYSEXEC マクロプログラムステートメントも、DATA ステップ実行中での使用は想定されていません。ただし CALL SYSTEM ルーチンは、DATA ステップ内からの実行が可能です。

注: X Window System の場合、引数のない X ステートメントはサポートされません。

### 比較

ウィンドウセッションでは、コマンドラインからコマンドを発行する点を除くと、X コマンドは X ステートメントと同じように動作します。X ステートメントは **プログラムエディタ** ウィンドウからサブミットします。

X ステートメントは、SYSTEM 関数、X コマンド、CALL SYSTEM ルーチンと同じように動作します。通常の場合、オーバーヘッドが少ないため、X ステートメント、X コマンド、または %SYSEXEC マクロステートメントを使用することをお勧めします。SYSTEM 関数は、条件を指定して実行されます。X ステートメントはグローバルステートメントです。DATA ステップのコンパイル時に実行されません。

### 関連項目:

["Executing Operating System Commands from Your SAS Session" \(Batch and Line Mode Processing in SAS Viya\)](#)

# 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 Viya Data Management and Utility Procedures Guide](#)
- [Learning SAS by Example: A Programmer's Guide](#)
- [Output Deliver System: The Basics and Beyond](#)
- [Batch and Line Mode Processing in SAS Viya](#)
- [SAS Viya コンポーネントオブジェクト: リファレンス](#)
- [SAS Viya Data Set Options: Reference](#)
- [SAS Viya Formats and Informats: Reference](#)
- [SAS Viya 関数と CALL ルーチン: リファレンス](#)
- [SAS Functions by Example](#)
- [SAS Viya 各国語サポート: リファレンスガイド](#)
- [SAS Viya ODS Graphics: Procedures Guide](#)
- [SAS Viya システムオプション: リファレンス](#)

SAS 刊行物の一覧については、[sas.com/store/books](https://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](mailto:sasbook@sas.com)  
Web アドレス: [sas.com/store/books](https://sas.com/store/books)



# キーワード

---

_ALL_CLEAR オプション	DATA ステートメント 48
CATNAME ステートメント 40	/STACK 引数
_ALL_LIST オプション	DATA ステートメント 48
CATNAME ステートメント 40	
_ALL_引数	~
FILENAME ステートメント 100	~(チルダ)フォーマット修飾子 204
LIBNAME ステートメント 227	~(チルダ)フォーマット修飾子, 定義
_ANY_オプション	208
WAITFOR ステートメント 415	
_BLANKPAGE_オプション, PUT ステートメント 318	@
_ERROR_変数 73	@(アットマーク)ラインホールド指定子, PUT ステートメント 327
_FILE_ 75	@@(ダブルアットマーク)ラインホールド指定子, PUT ステートメント 327
_FILE_=オプション	
FILE ステートメント 89	
_FILE_変数	
更新 91	
_INFILE_自動変数 146	&
_INFILE_=オプション	&(アンパサンド)フォーマット修飾子, 定義 208
INFILE ステートメント 146, 174	&(アンパサンド)フォーマット修飾子
PUT ステートメント 318	204
_IORC_自動変数	
MODIFY ステートメント 297	
_NULL_引数	%
DATA ステートメント 49	%INCLUDE ステートメント 138
_PAGE_オプション, PUT ステートメント 318	1つのカタログ内にある複数のエン
	トリ 145
;	引数 138
;(セミコロン), データ行内 40, 59	外部ファイルの挿入 144
	キーボード入力を含める 145
:	事前にサブミットされた行を挿入する 145
:(コロン)フォーマット修飾子 204	使用が求められる場合 143
:(コロン)フォーマット修飾子, 定義 208	詳細 143
	使用ルール 143
/	大量データの処理 376
/DEBUG 引数	データソース 143
DATA ステートメント 48	比較 144
/NESTING 引数	例 144
	%INC ステートメント 138
	%LIST ステートメント 270

%RUN ステートメント 368

## 1

1 対 1 のマージ 283

1 対 1 の読み込み 386, 388

## A

ABEND 引数

ABORT ステートメント 17

ABORT ステートメント 17

STOP ステートメントとの比較 393

引数 17

引数なし 17

詳細 19

比較 19

例 19

ACCESS=READONLY オプション

CATNAME ステートメント 40

ACCESS=オプション

LIBNAME ステートメント 227

ACTIVEMQ オプション

FILE ステートメント 77

ALTER 引数

DATA ステートメント 49

ALTER パスワード 52

ARRAY ステートメント 20

配列参照との比較, 明示的な 27

ATTRIB ステートメント 29

FORMAT ステートメントとの比較  
130

INFORMAT ステートメントとの比較  
178

LENGTH ステートメントとの比較  
224

## B

BATCHFILE オプション

FILENAME ステートメント、SFTP ア  
クセス方式 110

BLKSIZE=オプション

FILE ステートメント 80

INFILE ステートメント 149

BLOCKSIZE=オプション

FILENAME ステートメント、URL ア  
クセス方式 116

BY グループ

処理 36

BY-グループ処理

SET ステートメント 386

BY 値

重複, MODIFY ステートメント 295

BY グループ

開始と終了の識別 35

BY 処理 36

並べ替えを実行していないデータ  
37

BY ステートメント 33

DATA ステップ 35

PROC ステップ 36

SAS ビュー 36

引数 33

詳細 35

並べ替え順序の指定 37

例 36

BY 変数

指定 36

タイトルのカスタマイズ 405

## C

CALL ステートメント 39

CALL ルーチン

呼び出し 39

CANCEL 引数

ABORT ステートメント 17

CARDS4 ステートメント 40, 59

CARDS 引数

INFILE ステートメント 147

CARDS ステートメント 39, 57

CASE ステートメント 376

CATNAME ステートメント 40

引数 40

オプション 40

詳細 41

比較 42

例 42

catrefs 40

CD=オプション

FILENAME ステートメント、SFTP ア  
クセス方式 110

CHECKPOINT EXECUTE\_ALWAYS ス  
テートメント 43

CLEAR 引数

FILENAME ステートメント 100,  
104

LIBNAME ステートメント 227

CLEAR オプション

CATNAME ステートメント 40

COL=オプション

INFILE ステートメント 149

COLUMN=オプション

FILE ステートメント 80

INFILE ステートメント 149

Comment ステートメント 44

COMPRESS=オプション

LIBNAME ステートメント 227

COMPRESSION=オプション

FILENAME ステートメント、ZIP ア  
クセス方式 122



- CONNECT オプション
  - FILENAME ステートメント, URL アクセス方式 116
- CONTINUE ステートメント 46
  - LEAVE ステートメントとの比較 221
- CVP engine LIBNAME ステートメント 243
- CVP エンジン 245
  
- D**
- DATALINES4 59
- DATALINES4 ステートメント 40, 59
- DATALINES 引数
  - INFILE ステートメント 147, 167
- DATALINES ステートメント 39, 57
  - DATALINES4 ステートメントとの比較 59
- DATA ステートメント 47
  - DATA ステップビューの記述 54
  - DATA ステップビューの作成 53
  - 引数 47
  - 引数なし 47
  - カスタムレポートの作成 55
  - コンパイル済み DATA ステッププログラム
    - の作成 53
  - コンパイル済み DATA ステッププログラム
    - の実行 54
  - 出力データセットの作成 53
  - 詳細 52
  - データセットを作成しない場合 53
  - 入力 DATA ステップビューの作成 54
  - ネストレベルの表示 56
  - 例 54
- DATA ステップ
  - BY ステートメント 35
  - MODIFY ステートメント 298
  - 中止 17, 365, 393
- DATA ステップステートメント 1
  - グローバル、定義 3
  - 実行 1
  - 宣言 1
- DATA ステップの実行ステートメント 1
- DATA ステップの宣言ステートメント 1
- DATA ステップビュー
  - 記述 54
  - 作成 53
  - ソースコードの読み込み 61
- DATA ステッププログラム
  - コンパイル済み、実行 74
- DATA ステッププログラム, ソースコードの読み込み 61
- DATA 命名規則 47
- DEBUG オプション
  - FILENAME ステートメント, URL アクセス方式 117
  - FILENAME ステートメント, ZIP アクセス方式 122
  - FILENAME ステートメント, SFTP アクセス方式 110
- DEFAULT=引数
  - %INCLUDE ステートメント 141
  - INFORMAT ステートメント 176
  - LENGTH ステートメント 222
- DELETE ステートメント 60
  - DROP ステートメントとの比較 71
  - IF ステートメントとの比較, サブセット化 135
- DELIMITER=オプション
  - FILE ステートメント 80
  - INFILE ステートメント 149, 167
- DESCENDING 引数
  - BY ステートメント 33
- DESCRIBE ステートメント 61
- DIR オプション
  - FILENAME ステートメント, SFTP アクセス方式 110
- DISK オプション
  - FILE ステートメント 77
- DLM=オプション
  - INFILE ステートメント 149
- DLMSOPT=オプション
  - FILE ステートメント 81
  - INFILE ステートメント 150
- DLMSTR=オプション
  - FILE ステートメント 81
  - INFILE ステートメント 150
- DO UNTIL ステートメント 67
  - DO WHILE ステートメントとの比較 69
  - DO ステートメントとの比較 63
  - DO ステートメントとの比較, 反復 65
- DO WHILE ステートメント 69
  - DO UNTIL ステートメントとの比較 68
  - DO ステートメントとの比較 63
  - DO ステートメントとの比較, 反復 65
- DO-loops
  - DO ステートメント, 反復 63
- DO ステートメント 62
  - DO WHILE ステートメントとの比較 68, 69
- Do ステートメント, 反復
  - DO WHILE ステートメントとの比較 69
- DO ステートメント, 反復 63

DO UNTIL ステートメントとの比較 68  
 DO ステートメントとの比較 63  
 DO ループ  
 DO UNTIL ステートメント 67  
 DO WHILE ステートメント 69  
 DO ステートメント 62  
 GO TO ステートメント 132  
 再開 46, 221  
 終了 72  
 中止 46, 221  
 DO ループ処理  
 終了値 389  
 DROP=データセットオプション  
 DROP ステートメントとの比較 71  
 DROPOVER オプション  
 FILE ステートメント 81  
 DROP ステートメント 70  
 DELETE ステートメントとの比較 60  
 KEEP ステートメントとの比較 216  
 DSD オプション  
 FILE ステートメント 82  
 INFILE ステートメント 151, 168  
 DUMMY オプション  
 FILE ステートメント 77

**E**  
 ENCODING=オプション  
 FILENAME ステートメント 99, 108  
 FILENAME ステートメント, ZIP アクセス方式 122  
 FILE ステートメント 79, 95  
 INFILE ステートメント 151, 175  
 END=引数  
 MODIFY ステートメント 289  
 UPDATE ステートメント 408  
 END=オプション  
 INFILE ステートメント 151  
 SET ステートメント 378, 389  
 ENDSAS ステートメント 72  
 END ステートメント 72  
 ENGINE=システムオプション 238  
 EOF=オプション  
 INFILE ステートメント 152  
 EOV=オプション  
 INFILE ステートメント 152  
 ERROR ステートメント 73  
 EXECUTE ステートメント 74  
 EXPANDTABS オプション  
 INFILE ステートメント 152  
 EXTENDOBSCOUNTER=オプション  
 LIBNAME ステートメント 230

**F**  
 FILEEXT=オプション  
 FILENAME ステートメント, ZIP アクセス方式 122  
 FILELOCKS=オプション  
 LIBNAME ステートメント 234  
 FILENAME=オプション  
 FILE ステートメント 83  
 INFILE ステートメント 152  
 FILENAME ステートメント 96  
 LIBNAME ステートメントと~ 107  
 PUT ステートメントの出力先の指定 107  
 REDIRECT ステートメントとの比較 353  
 引数 97  
 エンコーディング指定 99  
 エンコードの指定 108  
 オプション 100  
 外部ファイルからの区切られたデータの読み込み 103  
 外部ファイルのファイル参照名 103, 106  
 集約記憶域のファイル参照名 107  
 出力デバイスのファイル参照名 104  
 詳細 103  
 定義 103  
 比較 106  
 ファイル参照名の関連付けを取り消し 100, 104  
 ファイル属性をログに書き込み 104  
 ファイルの属性をログに書き込み 100  
 例 106  
 ロックダウン 96  
 FILENAME ステートメント, URL アクセス方式 115, 121  
 URL オプション 115  
 URL ファイルの読み込み 120  
 Web サイトのファイルへのアクセス 120  
 ZIP オプション 121  
 引数 115  
 詳細 119  
 ユーザー ID とパスワード 120  
 例 120  
 FILENAME ステートメント, ZIP アクセス方式  
 引数 121  
 FILENAME ステートメント, SFTP アクセス方式 109  
 SFTP オプション 110  
 引数 109  
 詳細 113  
 比較 114  
 プロンプト 114

例 114  
 FILEVAR=オプション  
 FILE ステートメント 83, 94  
 INFILE ステートメント 152, 171  
 FILE 引数  
 ABORT ステートメント 17  
 FILE ステートメント 75  
 \_FILE\_ 変数の更新 91  
 長すぎる出力行 95  
 外部ファイル, 直接更新 90  
 改ページ 93  
 現在の出力ファイル 94  
 出力バッファ, コンテンツへのアクセス 91  
 出力ファイルのエンコーディング 95  
 詳細 90  
 新規ページでステートメントを実行する 92  
 比較 92  
 ページ全体のコンテンツを配置する 93  
 例 92  
 ロックダウン 75  
 FIRST. 変数 35  
 FIRSTOBS=オプション  
 INFILE ステートメント 153  
 FLOWOVER オプション  
 FILE ステートメント 83  
 INFILE ステートメント 153, 169  
 FOOTNOTES オプション  
 FILE ステートメント 84  
 FOOTNOTE ステートメント 125  
 引数なし 125  
 詳細 128  
 比較 128  
 例 128  
 FORMAT ステートメント 129

**G**

GO TO ステートメント 132  
 GOTO ステートメント 132  
 GROUPFORMAT 引数  
 BY ステートメント 33  
 GTERM オプション  
 FILE ステートメント 77

**H**

HEADER=オプション  
 FILE ステートメント 84, 92  
 HEADERS=オプション  
 FILENAME ステートメント, URL アクセス方式 117  
 HOST=オプション

FILENAME ステートメント, SFTP アクセス方式 111

**I**

I/O の制御  
 MODIFY ステートメント 308  
 IF, THEN/ELSE ステートメント 137  
 IF ステートメントとの比較, サブセット化 135  
 IF ステートメント, サブセット化  
 DELETE ステートメントとの比較 60  
 IF ステートメント, サブセット化 134  
 INDSNAME オプション  
 SET ステートメント 378  
 INENCODING=オプション  
 LIBNAME ステートメント 230  
 INFILE ステートメント 146  
 DBMS 指定 146  
 INPUT ステートメントとの比較 192  
 引数 146  
 エンコードの指定 175  
 オプション 146  
 外部ファイルの直接更新 160, 172  
 可変長レコード, スキャン 170  
 可変長レコード, 読み込み 170  
 行の終わりを越えて読み込む 165  
 区切り文字 167  
 区切り文字で区切られたデータ, 読み込み 161  
 欠損値, リスト入力 169  
 コピーされたレコードの切り捨て 172  
 詳細 160  
 動作環境オプション 146  
 長い入力ストリームデータレコードの読み込み 165  
 入力バッファ, コンテンツへのアクセス 160  
 入力バッファ, データ処理 173  
 比較 167  
 複数の入力ファイル 160, 171  
 ポインタ位置 172  
 短いレコード 169  
 例 167  
 ロックダウン 146  
 INFORMAT ステートメント 176  
 INPUT ステートメント 179  
 PUT ステートメントとの比較 329  
 カラム 197  
 名前付き 212  
 フォーマット 200  
 読み込むファイルの指定 146  
 INPUT ステートメント, カラム 197

- INPUT ステートメント, 名前付き 212  
 INPUT ステートメント, フォーマット 200  
 INPUT ステートメント, リスト 204  
   詳細 207  
   例 209  
 IOM クライアント  
   サブミットされた SAS プログラムの追跡 395
- J**  
 JMP Engine LIBNAME ステートメント 247  
 JMS オプション  
   FILE ステートメント 77  
 JSON Engine LIBNAME ステートメント 249
- K**  
 KEEP=データセットオプション  
   KEEP ステートメントとの比較 216  
 KEEP ステートメント 216  
   DROP ステートメントとの比較 71  
   RETAIN ステートメントとの比較 363  
 KEY=引数  
   MODIFY ステートメント 289  
 KEY=オプション  
   SET ステートメント 378, 388
- L**  
 LABEL ステートメント 217  
   ステートメントラベルとの比較 220  
 LAST. 変数 35  
 LEAVE ステートメント 46, 221  
   CONTINUE ステートメントとの比較 46  
 LENGTH=オプション  
   INFILE ステートメント 154, 170  
 LENGTH ステートメント 222  
 LIBNAME ステートメント 225  
   engine-host-options 225  
   FILENAME ステートメントと～ 107  
   引数 225  
   オプション 225  
   カタログの連結, 暗示的 240, 242  
   詳細 237  
   データセットの永久保存, 1 レベル名 242  
   データライブラリからライブラリ参照名の関連付けを取り消す 239  
   データライブラリ属性, ログに書き込む 239  
   データライブラリにライブラリ参照名を関連付ける 239  
   データライブラリの連結 239  
   データライブラリの連結, 論理的 241  
   比較 241  
   ライブラリ参照名の割り当て 241  
   ライブラリ連結ルール 240  
   例 241  
   ロックダウン 225  
   ～からのエンジン名の省略 238  
 LIBNAME ステートメント, CVP Engine 243  
 LIBNAME ステートメント, JMP Engine 247  
 LIBNAME ステートメント, JSON Engine 249  
 LIBNAME ステートメント, WebDAV サーバーアクセス 262  
 LINE=オプション  
   FILE ステートメント 84  
   INFILE ステートメント 154  
 LINES4 ステートメント 40, 59  
 LINESIZE=オプション  
   FILE ステートメント 84  
   INFILE ステートメント 154  
 LINESLEFT=オプション  
   FILE ステートメント 85, 93  
 LINES ステートメント 39, 57  
 LINK ステートメント 266  
   GO TO ステートメントとの比較 133  
 list output  
   スペース設定 345  
 LIST 引数  
   FILENAME ステートメント 100, 104  
   LIBNAME ステートメント 227  
 LIST オプション  
   CATNAME ステートメント 40  
 LIST ステートメント 268  
 LOCKDOWN ステートメント 274  
 LOCK ステートメント 271  
 LOG オプション  
   FILE ステートメント 76  
 LOSTCARD ステートメント 278  
 LOWCASE\_MEMNAME=オプション  
   FILENAME ステートメント, ZIP アクセス方式 123  
 LRECL=オプション  
   FILENAME ステートメント, URL アクセス方式 117  
   FILENAME ステートメント, ZIP アクセス方式 123  
   FILENAME ステートメント, SFTP アクセス方式 111

FILE ステートメント 85  
 INFILE ステートメント 154  
 LS=オプション  
 INFILE ステートメント 154  
 LSA オプション  
 FILENAME ステートメント、SFTP ア  
 クセス方式 111  
 LSFIL=オプション  
 FILENAME ステートメント、SFTP ア  
 クセス方式 111  
 LS オプション  
 FILENAME ステートメント、SFTP ア  
 クセス方式 111

**M**

MEMBER=引数  
 FILENAME, ZIP アクセス方式 121  
 MERGE ステートメント 280  
 UPDATE ステートメントとの比較  
 411  
 MGET オプション  
 FILENAME ステートメント、SFTP ア  
 クセス方式 112  
 MISSING=システムオプション  
 MISSING ステートメントとの比較  
 289  
 MISSING ステートメント 288  
 MISSEVER オプション  
 INFILE ステートメント 155, 169  
 MODIFY ステートメント 289  
 \_IORC\_自動変数 297  
 DATA ステップ 298  
 I/O の制御 308  
 SYSRC 自動呼び出しマクロ 297  
 インデックス付きの値によるダイレ  
 クトアクセス 296  
 オブザベーション番号によるダイレ  
 クトアクセス 296  
 重複する BY 値 295  
 重複するインデックス値 296, 306  
 順次アクセス 297  
 詳細 295  
 データセットオプション 300  
 比較 300  
 マッチングアクセス 295  
 例 301  
 MOD オプション  
 FILE ステートメント 86

**N**

N=オプション  
 FILE ステートメント 86, 93  
 INFILE ステートメント 155  
 NBYTE=オプション

INFILE ステートメント 156  
 NEW オプション  
 FILENAME ステートメント、SFTP ア  
 クセス方式 112  
 NOBS=オプション  
 MODIFY ステートメント 289  
 SET ステートメント 378, 389  
 NOLIST 引数  
 ABORT ステートメント 17  
 DATA ステートメント 51  
 NOTSORTED 引数  
 BY ステートメント 33  
 Null ステートメント 311

**O**

OBS=オプション  
 INFILE ステートメント 156  
 ODS (Output Delivery System)  
 タイトルとフットノートのカスタマ  
 イズ 405  
 ODS オプション  
 FILE ステートメント 87  
 OLD オプション  
 FILE ステートメント 86, 87  
 OPEN=オプション  
 SET ステートメント 378  
 OPTIONS=オプション  
 FILENAME ステートメント、SFTP ア  
 クセス方式 112  
 OPTIONSX=オプション  
 FILENAME ステートメント、SFTP ア  
 クセス方式 112  
 OPTIONS ステートメント 313  
 OUTENCODING=オプション  
 LIBNAME ステートメント 231  
 OUTPUT ステートメント 314  
 REMOVE ステートメントとの比較  
 354  
 REPLACE ステートメントとの比較  
 358  
 OUTREP=オプション  
 LIBNAME ステートメント 231  
 OVERPRINT オプション, PUT ステ  
 ートメント 318

**P**

PAD オプション  
 FILE ステートメント 88  
 INFILE ステートメント 156  
 PAGESIZE=オプション  
 FILE ステートメント 88  
 PAGE ステートメント 317  
 PASS=オプション

- FILENAME ステートメント, URL アクセス方式 117
  - PATH オプション
    - FILENAME ステートメント, SFTP アクセス方式 112
  - PGM=引数
    - DATA ステートメント 50
  - PIPE オプション
    - FILE ステートメント 78
  - PLOTTER オプション
    - FILE ステートメント 78
  - POINT=オプション
    - MODIFY ステートメント 289
    - SET ステートメント 378, 388
  - POINTOBS=オプション
    - LIBNAME ステートメント 232
  - PPASS=オプション
    - FILENAME ステートメント, URL アクセス方式 117
  - PRINTER オプション
    - FILE ステートメント 78
  - PRINT オプション
    - FILE ステートメント 76, 88
    - INFILE ステートメント 156
  - PROC ステップ
    - BY ステートメント 36
  - PROMPT オプション
    - FILENAME ステートメント, URL アクセス方式 118
  - PROXY=オプション
    - FILENAME ステートメント, URL アクセス方式 118
  - PUSER=オプション
    - FILENAME ステートメント, URL アクセス方式 118
  - PUTLOG ステートメント 350
  - PUT ステートメント 318
    - FILE ステートメント 75
    - INPUT ステートメントとの比較 192
    - LIST ステートメントとの比較 268
    - 出力ファイル 75
    - ～の出力先の指定 107
  - PUT ステートメント, カラム 336
  - PUT ステートメント, 名前付き 347
  - PUT ステートメント, フォーマット 338
  - PUT ステートメント, リスト 343
    - 引数 343
    - 修飾リスト出力, 値の書き込み 347
    - 修飾リスト出力とフォーマット出力の違い 345
    - 詳細 345
    - 比較 345
    - 変数の書き込み 346
    - 文字列の書き込み 346
  - リスト出力 345
  - リスト出力, 値の書き込み 346
  - リスト出力, スペース設定 345
  - 例 346
  - PW 引数
    - DATA ステートメント 50
- ## R
- RCFM=オプション
    - FILENAME ステートメント, ZIP アクセス方式 123
  - READ 引数
    - DATA ステートメント 50
  - READ パスワード 52
  - RECFM=オプション
    - FILENAME ステートメント 100
    - FILENAME ステートメント, URL アクセス方式 118
    - FILENAME ステートメント, SFTP アクセス方式 112
    - FILE ステートメント 89
    - INFILE ステートメント 156
  - REDIRECT ステートメント 352
    - 引数 352
    - 例 353
  - REMOVE ステートメント 354
    - OUTPUT ステートメントとの比較 315
    - REPLACE ステートメントとの比較 354, 358
  - RENAME=データセットオプション
    - RENAME ステートメントとの比較 356
  - RENAME ステートメント 356
  - REPEMPTY=オプション
    - LIBNAME ステートメント 233
  - REPLACE ステートメント 357
    - OUTPUT ステートメントとの比較 315
    - REMOVE ステートメントとの比較 354
  - RESETLINE ステートメント 360
  - RETAIN ステートメント 361
    - KEEP ステートメントとの比較 216
    - 合計ステートメントとの比較 395
  - RETURN 引数
    - ABORT ステートメント 17
  - RETURN ステートメント 365
    - GO TO ステートメントとの比較 133
  - RSUBMIT ステートメント 399
  - RUN ステートメント 367

**S**

S2=引数  
 %INCLUDE ステートメント 141  
 SAS OPTIONS ウィンドウ、OPTIONS  
 ステートメントの比較 314  
 SAS/CONNECT  
 非同期プロセス 399  
 SASFILE ステートメント 369  
 SAS システムオプション  
 値の変更 313  
 SAS ジョブ  
 中止 17  
 SAS ジョブ, 終了 72  
 SAS ステートメント 1  
 SAS セッション  
 オペレーティングシステムコマンド  
 の発行 423  
 実行の中断 415  
 終了 72  
 中止 17  
 SAS データセット  
 オブザベーションの削除 354  
 書き込み 314  
 リダイレクト 352  
 SAS ビュー  
 BY ステートメント 36  
 SAS プログラム  
 ステートメントやデータ行の読み込  
 み 138  
 追跡、IOM クライアント 395  
 SAS ログ  
 新しいページにスキップ 317  
 ログ入力 268  
 SCANOVER オプション  
 INFILE ステートメント 157, 170  
 SELECT グループ、IF、THEN/ELSE ステ  
 ートメントとの比較 137  
 SELECT ステートメント 375  
 SELECT グループ内の WHEN ステ  
 ートメント 375  
 比較 376  
 例 376  
 SESSREF=引数  
 DATA ステートメント 51  
 SESSUID=引数  
 DATA ステートメント 51  
 SET ステートメント 378  
 1 対 1 の読み込み 386, 388  
 BY-グループ処理 386  
 INPUT ステートメントとの比較  
 192  
 MERGE ステートメントとの比較  
 284  
 引数 378  
 オブザベーションのマージ 387  
 オプション 378

サブセットの読み込み 388  
 詳細 384  
 データセットのインタリーブ 386,  
 387  
 データセットの結合 386  
 データセットの連結 386, 387  
 テーブルルックアップ 388  
 比較 386  
 例 387  
 SFTP アクセス方式  
 参照項目: FILENAME ステートメン  
 ト、SFTP アクセス方式  
 SFTP 引数  
 FILENAME ステートメント、SFTP ア  
 クセス方式 109  
 SFTP オプション  
 FILE ステートメント 78  
 SHAREBUFFERS オプション  
 INFILE ステートメント 157, 172  
 SHAREBUFS オプション  
 INFILE ステートメント 157  
 SKIP ステートメント 392  
 SOURCE=引数  
 DATA ステートメント 50  
 SSHD サーバー  
 非標準ポートの接続 114  
 START=オプション  
 INFILE ステートメント 158  
 STOPOVER オプション  
 FILE ステートメント 89  
 INFILE ステートメント 158, 169  
 STOP ステートメント 393  
 SUM 関数  
 合計ステートメントとの比較 395  
 SYSECHO ステートメント 395  
 SYSRC 自動呼び出しマクロ  
 MODIFY ステートメント 297  
 SYSRC マクロ変数  
 SYSTASK ステートメントのリターン  
 コード 399  
 WAITFOR ステートメントのリター  
 ンコード 416  
 SYSTASK ステートメント 396

**T**

TAPE オプション  
 FILE ステートメント 78  
 TEMP オプション  
 FILE ステートメント 78  
 TERMINAL オプション  
 FILE ステートメント 78  
 TERMSTR=オプション  
 FILENAME ステートメント、URL ア  
 クセス方式 119



FILENAME ステートメント, ZIP ア  
 クセス方式 124  
 FILE ステートメント 89  
 THREADS=  
 DATA ステートメント 51  
 TIMEOUT=オプション  
 WAITFOR ステートメント 415  
 TITLES オプション  
 FILE ステートメント 89  
 TITLE ステートメント 399  
 TLS プロトコル  
 FILENAME ステートメント, URL ア  
 クセス方式 119  
 TO ステートメント, LINK ステートメ  
 ントとの比較 267  
 Transport Layer Security(TLS)プロト  
 コル  
 FILENAME ステートメント, URL ア  
 クセス方式 119  
 TRUNCCOVER オプション  
 INFILE ステートメント 158, 170

**U**

UNBUFFERED オプション  
 INFILE ステートメント 159  
 UNBUF オプション  
 FILE ステートメント 89  
 INFILE ステートメント 159  
 UNIQUE=オプション  
 MODIFY ステートメント 289  
 UNIQUE オプション  
 SET ステートメント 378  
 UPDATEMODE=引数  
 UPDATE ステートメント 408  
 UPDATEMODE=オプション  
 MODIFY ステートメント 289  
 UPDATE ステートメント 408  
 MERGE ステートメントとの比較  
 284  
 UPRINTER オプション  
 FILE ステートメント 78  
 URL アクセス方式  
 参照項目: [xisError - indexSee](#)  
 primary entry "FILENAME ステ  
 ートメント、URL アクセス方式"  
 not found  
 URL オプション  
 FILE ステートメント 79  
 USER=オプション  
 FILENAME ステートメント, URL ア  
 クセス方式 119  
 FILENAME ステートメント, SFTP ア  
 クセス方式 113

**V**

VIEW=引数  
 DATA ステートメント 49

**W**

WAIT\_MILLISECONDS=オプション  
 FILENAME ステートメント、SFTP ア  
 クセス方式 113  
 WAITFOR ステートメント 415  
 WEBDAV 262  
 Web サイト  
 ファイルへのアクセス 120  
 WHEN ステートメント 375  
 SELECT グループ内 375  
 WHERE ステートメント 416  
 IF ステートメントとの比較, サブセ  
 ット化 135

**X**

X コマンドと X ステートメントの比較  
 424  
 X ステートメント 423

**Z**

ZIP アクセス方式  
 参照項目: [xisError - indexSee](#)  
 primary entry "FILENAME ステ  
 ートメント、ZIP アクセス方式"  
 not found

**あ**

アットマーク(@)引数  
 INPUT ステートメント 181  
 INPUT ステートメント, カラム入力  
 197  
 INPUT ステートメント, 名前付き入  
 力 212  
 INPUT ステートメント, フォーマッ  
 ト入力 200  
 PUT ステートメント 318  
 PUT ステートメント, カラム出力  
 336  
 PUT ステートメント, 名前付き出力  
 347  
 PUT ステートメント, フォーマット  
 出力 338  
 アットマーク(@)カラムポインタコン  
 トロール  
 INPUT ステートメント 182  
 PUT ステートメント 318  
 アットマーク(@)ラインホールド指定  
 子



PUT ステートメント 327  
 PUT ステートメント, カラム出力 327  
 アットマーク(@@)引数  
 INPUT ステートメント, カラム入力 197  
 INPUT ステートメント, フォーマット入力 200  
 PUT ステートメント, カラム出力 336  
 PUT ステートメント, 名前付き出力 347  
 アットマーク(@@)引数  
 PUT ステートメント, フォーマット出力 338  
 アンパサンド(&)フォーマット修飾子 204, 208  
 位置が揃っていないデータ 209  
 インターフェイスライブラリエンジン 237  
 インデックス  
   重複する値 296, 306  
 インデックス付きの値  
   ダイレクトアクセス 296  
 埋め込みのブランク  
   文字データ 210  
 エラーメッセージ  
   書き込み 73  
 エンコーディング  
   出力ファイル 95  
 エンジン  
   CVP エンジン 245  
 オブザベーション  
   SET ステートメントを用いた読み込み 378  
   書き込み 314  
   グループ化 37  
   削除 60, 354  
   サブセットの読み込み 388  
   置換 357  
   ドロップ 70  
   フォーマット値を用いてグループ化する 37  
   複数の結合 37  
   複数レコード 190  
   変更 289, 301  
   変更, インデックスにより検索 305  
   変更, オブザベーション番号により検索 304  
   変更, 他の SAS データセットへの書き込み 310  
   変更, トランザクションデータセットを使用 302  
   マージ 280, 387  
 オブザベーション, 選択  
   IF, THEN/ELSE ステートメント 137

IF, サブセット化 134  
 WHERE ステートメント 416  
 オブザベーションのグループ化 37  
   フォーマット値 37  
 オブザベーションのマージ 387  
 オペレーティングシステムコマンド  
   SAS セッションから発行 423

**か**

外部ファイル  
   エンコーディング指定 99  
   エンコードの指定 108  
   挿入 144  
   直接更新 90, 160, 172  
   定義 103  
   ファイル参照名の関連付けを取り消し 100  
   読み込むファイルの指定 146  
   ~からの区切られたデータの読み込み 103  
   ~からファイル参照名の関連付けを取り消し 104  
   ~属性をログに書き込み 104  
   ~にファイル参照名を関連付け 103, 106  
   ~の属性をログに書き込み 100

改ページ  
 現在のページに残っている行数に基づき決定する 93  
 ステートメントの実行 92

カタログ  
 %INCLUDE ステートメントと1つのカタログ内にある複数のエントリ 145  
 連結 40  
 連結, 暗示的 240, 242

カタログの連結  
 CATNAME ステートメント 40  
 暗示的 240, 242  
 ネストされたカタログ連結 43  
 ルール 41  
 論理的に連結されたカタログ 42

可変長レコード  
 文字列のスキャン 170  
 読み込み 170

カラム出力 324, 336  
 カラム入力 185, 197  
 カラムポインタコントロール  
   INPUT ステートメント 179  
   PUT ステートメント 318  
 カンマで区切られたデータ 209, 210  
 キーボード入力  
   含める 145  
 疑問符(?)フォーマット修飾子  
   INPUT ステートメント 185

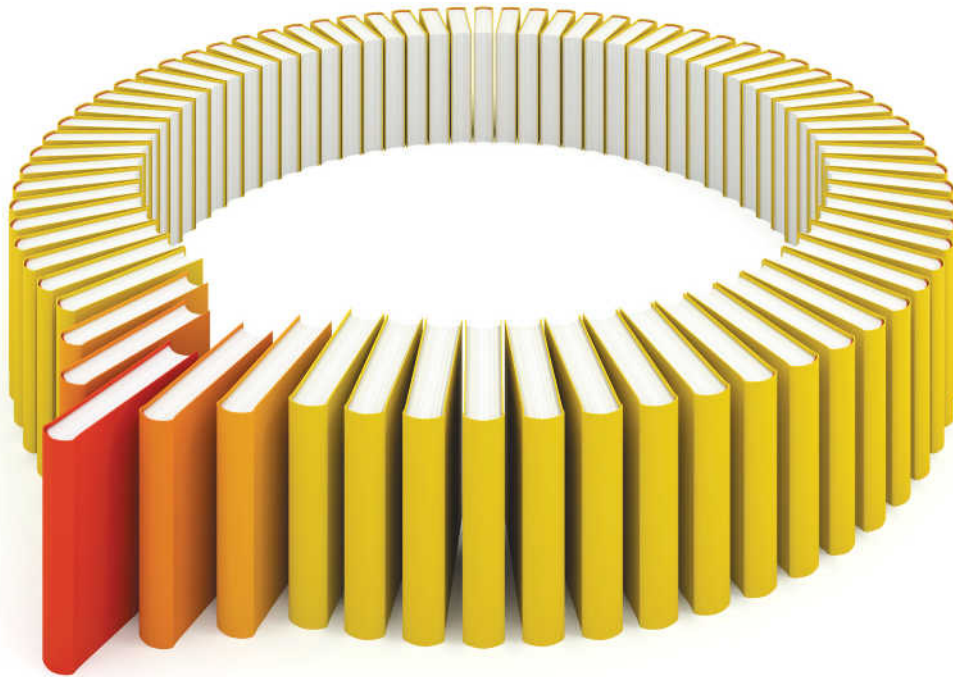
- 疑問符(?)フォーマット修飾子
  - INPUT ステートメント 184
- 行の終わりを越えて読み込む 165
- 行ポインタコントロール
  - INPUT ステートメント 179
  - PUT ステートメント 318
- 切り捨て
  - コピーされたレコード 172
- 区切り文字
  - INFILE ステートメント 167
- 区切り文字が区別されるデータ
  - FILE ステートメント 75
- 区切り文字で区切られたデータ
  - 読み込み 161
- 区切られたデータ 211
  - 外部ファイルからの~の読み込み 103
- グローバル DATA ステップステートメント
  - 定義 3
- 欠損値
  - MISSING ステートメント 288
  - 外部ファイルの読み込み 169
  - 入力 288
  - 文字で置き換え 288
  - リスト入力 169
- 欠損レコード, 入力 278
- 合計式 394
- 合計ステートメント 394
- 後置@
  - INPUT ステートメント, リスト 204
- 後置@@
  - INPUT ステートメント, リスト 204
- コピーされたレコード
  - 切り捨て 172
- コメント 44
- コロン(:)フォーマット修飾子 204, 208
- コンパイル済み DATA ステッププログラム
  - 作成 53
  - 実行 54, 74
  - ソースコードの読み込み 61
  - パスワード 56
- フォーマット出力との違い 345
- 修飾リスト入力 208
  - 区切られたデータの読み込み 211
  - フォーマット入力との違い 208
- 集約記憶域
  - ~のファイル参照名 107
- 出力
  - カラム 324, 336
  - 名前付き 325, 347
  - フォーマット 325, 338
  - フットノート 125
  - リスト 325
- 出力形式
  - 変数への関連付け 129
- 出力データセット 53
  - 作成 53
  - リダイレクト 352
- 出力デバイス
  - ~にファイル参照名を関連付け 104
- 出力バッファ
  - コンテンツへのアクセス 91
- 出力ファイル
  - PUT ステートメント 75
  - エンコーディング 95
  - 長すぎる出力行 95
  - 現在のファイルの識別 94
  - 現在のファイルを動的に変更する 94
- 順次アクセス
  - MODIFY ステートメント 297
- ステートメント 1
  - DATA ステップステートメント 1
  - 改ページ時に実行する 92
  - 実行 1
  - 宣言 1
  - ステートメントラベル 219
  - ステートメントラベル, ジャンプ 266
  - スラッシュ(/)行ポインタコントロール
    - INPUT ステートメント 184
    - PUT ステートメント 318
  - セミコロン(;), データ行内 40, 59
  - 宣言ステートメント 1
- た
- タイトル
  - BY 変数を用いたカスタマイズ 405
  - ODS を用いたカスタマイズ 405
- ダイレクトアクセス
  - インデックス付きの値 296
  - オブザベーション番号 296
- ダブルアットマーク(@@)引数
  - INPUT ステートメント 181
  - INPUT ステートメント, 名前付き入力 212
  - PUT ステートメント 318
- さ
- 式、合計 394
- 実行ステートメント 1
- 実行の中断 415
  - SAS セッション 415
- 実行の停止
  - SAS セッション 415
- 修飾リスト出力 345
  - :を用いた値の書き込み 347
  - ~を用いた値の書き込み 347

- ダブルアットマーク(@@)ラインホールド指定子, PUT ステートメント 327
- 端末
  - ～のファイル参照名 104
- チェックポイント-再開モード 44
- チルダ(~)フォーマット修飾子 204, 208
- データ行
  - 読み込み 39, 59, 138
- データセット
  - 関連項目: DATA ステートメント
  - 1対1の読み込み 386, 388
  - インタリーブ 386, 387
  - 永久保存, 1レベル名 242
  - オブザベーションの読み込み 378, 387
  - オブザベーションの読み込み, 複数回 387
  - 結合 386
  - 連結 386, 387
- データセットオプション
  - MODIFY ステートメント 300
- データセットのインタリーブ
  - SET ステートメント 386, 387
- データセットのリダイレクト 352
- データセットの連結
  - SET ステートメント 386, 387
- データセットリスト
  - MERGE ステートメント 280
  - SET ステートメント 378
- データライブラリ
  - 属性をログに書き込む 239
  - ライブラリ参照名の関連付けを取り消す 239
  - ライブラリ参照名を関連付ける 239
  - 連結 239
  - 連結, 論理的 241
- データライブラリの連結 239
  - 論理的 241
- テーブルルックアップ
  - マスタファイルの重複するオブザベーション 388
- トランザクションデータセット
  - オブザベーションの変更 302
- ドル記号(\$)引数
  - INPUT ステートメント 180
  - INPUT ステートメント, カラム入力 197
  - INPUT ステートメント, 名前付き入力 212
  - LENGTH ステートメント 222
- 名前付き出力 325, 347
- 名前付き入力 186, 212
- 並べ替え順序
  - BY ステートメントを用いた指定 37
- 入力
  - カラム 185, 197
  - 欠損値 288
  - 欠損レコード 278
  - 現在のセッションでのリスト 268
  - 再同期 278
  - データ終了インジケータ 311
  - 名前付き 186, 212
  - 入力 186
  - フォーマット 186, 200
  - フォーマット方法の記述 179
  - 変数への割り当て 179
  - 無効なデータ 191, 278
  - リスト入力 204
  - ログ 268
- 入力 DATA ステップビュー
  - 作成 54
- 入力カラム 200
- 入力形式
  - 位置が揃っていないデータの読み込み 210
  - 変数への関連付け 176
- 入力ストリームデータ
  - 長いレコードの読み込み 165
- 入力データ
  - 行の終わりを超えて読み込む 165
- 入力データセット
  - リダイレクト 352
- 入力バッファ
  - アクセス, 複数のファイル 174
  - コンテンツへのアクセス 160
  - データ処理 173
- 入力ファイル
  - コピーされたレコードの切り捨て 172
  - 複数のファイルの読み込み 160, 171
- ネイティブライブラリエンジン 237
- ネストされたカタログ連結 43
- ネストレベル
  - 表示 56
- は
- 配列
  - 書き込み 329
  - 処理する要素の記述 25
  - 要素を定義 20
- 配列参照 25
- 配列参照, 明示的な 25
  - ARRAY ステートメントとの比較 23
- 配列参照ステートメント 25
- パスワード

- ALTER 52
  - DATA ステップ 52
  - READ 52
  - コンパイル済み DATA ステッププログラム 56
  - バッチ処理
    - チェックポイント-再開モード 44
  - バッファ, 割り当て
    - SASFILE ステートメント 369
  - ビューエンジン 237
  - ファイル, マスター
    - 更新 408
  - ファイル参照名
    - FILENAME ステートメント 96
    - 外部ファイルからの関連付けを取り消し 100
    - 外部ファイルから～の関連付けを取り消し 104
    - 外部ファイルに～を関連付け 103, 106
    - 出力デバイスに～を関連付け 104
    - 定義 103
    - ～を集約記憶域に関連付け 107
  - フォーマット出力 325, 338
  - フォーマット入力 186, 200
    - 修飾リスト入力との違い 208
  - フットノート
    - ODS を用いたカスタマイズ 405
  - プラス記号(+)カラムポインタコントロール
    - INPUT ステートメント 183
    - PUT ステートメント 318
  - プリンタ
    - ～のファイル参照名 104
  - プログラミングステートメント
    - 読み込み 138
  - プログラミングステートメントとデータ行の読み込み 138
  - プロシジャ出力
    - フットノート 125
  - プロッタ
    - ～のファイル参照名 104
  - ページサイズ
    - 列が 2 列配置される形式 93
  - 変数
    - \_ERROR\_ 設定 73
    - BY 変数 36
    - FIRST. 35
    - LAST. 35
    - 出力形式の関連付け 129
    - 長さ、指定 222
    - 名前の変更 356
    - 入力形式の関連付け 176
    - 入力の割り当て 180
    - 変数の保持 361
    - ラベル付け 217
  - ポインタ位置 172
  - ポインタコントロール
    - INPUT ステートメント 187
    - PUT ステートメント 326
  - ポンド記号(#)行ポインタコントロール
    - INPUT ステートメント 184
    - PUT ステートメント 318
- ま**
- マスターファイル, 更新 408
  - マッチマージ 283
  - マッチングアクセス 295
  - 短いレコード 169
  - メッセージ
    - ログへの書き込み 350
  - 文字データ
    - 埋め込みのブランク 210
  - 文字変数埋め込み(CVP)エンジン 245
- や**
- ユニバーサルプリンタ
    - ～のファイル参照名 104
- ら**
- ライブラリエンジン 237
  - ライブラリ参照名
    - データライブラリから関連付けを取り消す 239
    - データライブラリに関連づける 239
    - 割り当て 241
  - ライブラリ連結ルール 240
  - ラインホールド指定子
    - INPUT ステートメント 188
    - PUT ステートメント 327
  - ラベル
    - ステートメントラベル 219
  - ラベル: ステートメント 219
  - リスト出力 325, 345
    - 関連項目: xisError - indexSee primary entry "PUT ステートメント、リスト" not found
    - :を用いた値の書き込み 346
    - PUT ステートメント, リスト 343
  - リスト入力 186, 204
    - 位置が揃っていないデータの読み込み 209
    - 引用符を含むデータ 209
    - 埋め込みのブランクを含む文字データ 210
    - カンマ区切りデータ 210
    - 区切られたデータの読み込み 211
    - 欠損値 169

- 修飾 208, 211
- 使用する場合 207
- 単純 207, 209
- 入力形式を使用した位置が揃っていないデータの読み込み 210
- リモートファイル
  - SFTP アクセス方式 109
  - URL アクセス方式 115
  - ZIP アクセス方式 121
- リモートホスト
  - ディレクトリからファイルを読み込む 115
- 列
  - 列が 2 列配置されるページ形式 93
  - 列が 2 列配置される形式 93
- レポート
  - DATA ステートメントを用いた作成 55
- ログ
  - 外部ファイル属性を~に書き込み 104
  - 外部ファイルの属性を~に書き込み 100
  - データライブラリ属性を書き込む 239
  - メッセージの書き込み 350
- ロックダウン
  - FILENAME ステートメント 96
  - FILE ステートメント 75
  - INFILE ステートメント 146
  - LIBNAME ステートメント 225
- わ
  - 割り当てステートメント 28





# 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](http://support.sas.com/bookstore)  
for additional books and resources.

  
THE POWER TO KNOW.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613

