THE POWER TO KNOW.

SAS® 9.4 言語リファレンス
解説編
第 4 版

SAS® ドキュメント

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2014. *SAS® 9.4 言語リファレンス:解説編(第 4 版)*. Cary, NC: SAS Institute Inc.

SAS® 9.4 言語リファレンス:解説編(第 4 版)

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

October 2014

目次

新機能: 9.4 Base SAS 言語リファレンス:解説編

概要

SAS 9.4 では、次の変更と拡張が行われました。

- SAS 9.4 の 3 番目のメンテナンスリリースで、新しい Avenir Next TrueType フォントを追加
- SAS 9.4 の最初のメンテナンスリリースで、LOCKDOWN ステートメントおよび LOCKDOWN システムオプションを追加
- SAS 9.4 の 2 番目のメンテナンスリリースで、LOCKDOWN 機能を強化
- SAS 9.4 の 2 番目のメンテナンスリリースでは、ユニバーサルプリンタでフォントの斜体化と太字化をサポート
- ユニバーサル印刷 における、グラフィック出力タイプの追加、ならびに GIF および SVG ファイルのアニメーションのサポートを拡張
- SAS DATA ステップビューでのバッファサイズ指定をサポート
- 新しい多言語およびアジア単一言語の TrueType フォントをサポート
- SAS データセットおよび変数での拡張属性 をサポート
- 32 ビット SAS データファイルのオブザベーションカウント を拡張する機能強化
- SAS データファイルの保護を拡張"データファイルの保護"
- VIEWTABLE の 列見出し機能を拡張

SAS System の機能

ユニバーサル印刷

- 複数ページの GIF 画像および SVG ファイルをアニメーション表示することができます。
- TIFF 画像、ならびに EMFPlus および EMFDual メタファイル形式を作成できるようになりました。
- EMF ユニバーサルプリンタ、および PostScript ユニバーサルプリンタを使用して印刷される GIF 画像では、透過性がサポートされます。

- COLOPHON=システムオプションを使用して、ユニバーサル印刷出力に表示されないプリンタマークを追加できます。
- SVGMAGNIFYBUTTON システムオプションを設定すると、SVG ドキュメントを拡大できます。SVG ドキュメントの作成時に、ドキュメントに拡大ツールが埋め込まれます。

“ユニバーサル印刷を使用した TIFF 画像の作成” (327 ページ)を参照してください。

フォントの斜体化と太字化をサポート

SAS 9.4 の 2 番目のメンテナンスリリースでは、斜体や太字体をサポートしていないユニバーサルプリンタフォントに対して斜体や太字体を指定した場合、デフォルトでそれらのフォントが斜体や太字体で表示されるようになりました。“フォントの斜体化と太字化” (278 ページ)を参照してください。

TrueType フォント

SAS 9.4 では、次の新しいフォントとフォントの置換がサポートされるようになりました。

- Avenir Next TrueType フォント(SAS 9.4 のメンテナンスリリース 3)
- Monotype Sans WT (J、K、SC、TC)フォントが、新しい Arial Unicode MS フォントに置換されます。
- Thorndale Duospace WT (J、K、SC、TC)フォントが、新しい Times New Roman Uni フォントに置換されます。
- Sim Hei、SimSun および NSimSun が、CSongGB18030C-Light、CSongGB18030C-LightHWL、MYingHei_18030_C-Medium、MYingHei_18030_C-MediumHWL に置換されます。

“SAS 提供の TrueType フォント” (271 ページ)を参照してください。

Avenir Next TrueType フォント

SAS 9.4 のメンテナンスリリース 3 では現在 Avenir Next TrueType フォントが使用可能です。Avenir Next は、画面表示用に設計された現代的な書体を特徴とするサンセリフの TrueType フォントファミリです。この原型は主に Futura および Univers 書体です。ラテンおよびキリル文字セットにおいて SAS 9.4 のメンテナンスリリース 3 で使用可能な新しい Avenir Next フォントのリストを次に示します。

ラテン	キリル
フォント名	フォント名
Avenir Next LT W04 Demi	Avenir Next Cyr W04 Demi
Avenir Next LT W04 Demi Italic	Avenir Next Cyr W04 Demi Italic
Avenir Next LT W04 Italic	Avenir Next Cyr W04 Italic Regular
Avenir Next LT W04 Light Italic	Avenir Next Cyr W04 Light It
Avenir Next LT W04 Light	Avenir Next Cyr W04 Light
Avenir Next LT W04 Regular	Avenir Next Cyr W04 Regular

SAS DATA ステップビューバッファのサポート

DATA ステップビューに使用されるバッファのサイズを指定できるようになりました。ビューバッファを設定して実行を高速化すると、より多くの生成オブザベーションを保持できるようになり、タスクの切り替えが少なくてすみます。“VBUFSIZE=” (191 ページ)を参照してください。

拡張属性

拡張属性を使用することで、変数やデータセット用のカスタマイズ属性を作成できるようになりました。拡張属性とは、お使いの SAS ファイル用のカスタマイズされたメタデータのことです。これらはユーザー定義属性であり、SAS データセットや変数と関連付けることができます。“拡張属性” (655 ページ)を参照してください。

オブザベーションカウントの拡張

SAS 9.4 では、オブザベーションカウントを拡張する機能が強化されており、拡張オブザベーションカウント付きの 32 ビット SAS データファイルを自動的に作成できるほか、EXTENDOBSCOUNTER=システムオプションが提供されています。SAS 9.4 では、EXTENDOBSCOUNTER=データセットオプション、LIBNAME ステートメントオプション、およびシステムオプションは、デフォルトで YES に設定されています。“32 ビット SAS データファイルのオブザベーションカウントの拡張” (658 ページ)を参照してください。

CEDA(クロス環境データアクセス)

クロス環境データアクセス(CEDA)では、拡張属性の読み込みや更新が行えません。

SMTP 対応電子メールの認証プロトコル

SMTP 対応電子メールのサーバー認証プロトコルが、EMAILHOST=システムオプションで指定されたユーザー ID を検索するよう拡張されました。

LOCKDOWN 状態にある場合の制限

SAS 9.4 の最初のメンテナンスリリースで、LOCKDOWN ステートメントと LOCKDOWN システムオプションが新しく追加されました。LOCKDOWN を使用すると、クライアント/サーバー環境で実行(たとえば、SAS Enterprise Guide を使用)している場合、SAS サーバー管理者が、SAS クライアントからアクセスするディレクトリおよびファイルの環境を作成できます。また、ディレクトリやファイルに関する制限が存在することに加えて、SAS がロックダウン状態になると、一部の言語要素が利用できなくなります。

“サーバーがロックダウン状態になった場合の SAS 処理の制限” (17 ページ)を参照してください。

LOCKDOWN ステートメントの拡張

SAS 9.4 の 2 番目のメンテナンスリリースでは LOCKDOWN ステートメントが拡張されており、SAS セッションがロックダウン状態になると、特定のアクセス方式およびそれらの関連プロシジャが無効になります。

また、SAS 9.4 の 2 番目のメンテナンスリリースでは、LOCKDOWN ステートメントの ENABLE_AMS=オプションが追加されました。このオプションを使うことで、管理者は、LOCKDOWN が有効である場合に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。“サーバーがロックダウン状態になった場合の SAS 処理の制限” (17 ページ)を参照してください。

データファイルの保護

SAS 9.4 では、Advanced Encryption Standard (AES)暗号化をサポートするようになりました。AES 暗号化では、キー値を使用してより強力な暗号化を実現できます。“AES 暗号化” (732 ページ)を参照してください。

SAS 9.4 では、データセキュリティを強化するために、メタデータ連結ライブラリをサポートするようになりました。メタデータ連結ライブラリは、対応するメタデータ保護テーブルオブジェクトに結び付けられる物理的なライブラリです。“メタデータ連結ライブラリ” (735 ページ)を参照してください。

VIEWTABLE の列見出し

SAS 9.4 の最初のメンテナンスリリースでは、VIEWTABLE で表示しているデータセットの列ラベルまたは列名を保存できるようになりました。

ユーザー補助

この製品のユーザー補助の詳細については、support.sas.com にある Accessibility Features of the SAS Windowing Environment を参照してください。

1 部

SAS System の概念

1 章

Base SAS の基本概念

SAS について

SAS は、企業全体のビジネスユーザーに向けたソリューションのセットであり、次のようなタスクを実行するための強力な第 4 世代のプログラミング言語を提供します。

- データの入力、検索、管理
- レポートやグラフィックの作成
- 統計および数理解析

- ビジネスの計画、予測、意思決定支援
- オペレーションズリサーチおよびプロジェクト管理
- 品質管理
- アプリケーション開発

Base SAS ソフトウェアをファンデーションとして使用すると、SAS と多数の SAS ビジネスソリューションを統合できるため、大規模な業務機能の実行が可能になります。これには、データウェアハウス、データマイニング、人的資源管理や財務管理における意思決定支援などが含まれます。

Base SAS の概要

SAS System の基本プロダクトは、Base SAS ソフトウェアです。Base SAS ソフトウェアは、次に示す SAS System の基本機能を提供します。

DATA ステップ
: データを操作および管理するために使用するプログラミング言語

SAS プロシジャ
: データ分析やレポート作成を実行するためのソフトウェアツール

マクロ機能
: SAS プログラムを拡張してカスタマイズを行うため、プログラムのコード量を減らすためのツール

DATA ステップデバッガ
: DATA ステッププログラム内の論理的な問題を検出するのに役立つデバッグツール

Output Delivery System (ODS)
: SAS データセット、プロシジャ出力ファイル、ハイパーテキストマークアップ言語 (HTML)など、さまざまな形式で出力を配信するシステム

SAS ウィンドウ環境
: SAS プログラムの実行やテストを簡単に行える対話型のグラフィカルユーザーインターフェイス

本書では、SAS 言語の概念についてのみ説明します。Base SAS ソフトウェア機能の完全なガイドについては、*SAS Output Delivery System: User's Guide*、*SAS National Language Support (NLS): Reference Guide*、*Base SAS Procedures Guide*、*SAS XML LIBNAME Engine: ユーザーガイド*、*SAS Macro Language: Reference*、*SAS Logging: Configuration and Programming Reference*、およびオンラインチュートリアルの SAS 入門ガイドを参照してください。SAS ウィンドウ環境の詳細については、オンラインヘルプを参照してください。

SAS 言語のコンポーネント

SAS ファイル

SAS System で作業を行う場合、SAS System によって作成および管理されるファイル、または動作環境によって作成および管理される SAS System と関連付けられていないファイルを使用します。.SAS System で使用できるファイルのうち、SAS System によっ

て識別される形式または構造を持つファイルは、SAS ファイルと呼びます。SAS ファイルはすべて、SAS ライブラリ内に存在します。

最もよく使用される SAS ファイルは、SAS データセットです。SAS データセットは、SAS System が処理できる形式のデータファイルです。もう 1 つの一般的な SAS ファイルは、SAS カタログです。SAS カタログには、SAS ジョブで使用されるさまざまな種類の情報が格納されています。例としては、データ値の読み込みや出力を行う命令や、SAS ウィンドウ環境で使用するファンクションキーの設定などが挙げられます。ストアド SAS プログラムは SAS ファイルの種類の 1 つで、繰り返し使用するために作成保存されたコンパイル済みプログラムファイルを含みます。

動作環境の情報
SAS ライブラリの実装形式は、動作環境によって、ファイル間の物理的な関係を示す場合も、論理的な関係を示す場合もあります。SAS ライブラリの特性の詳細については、使用している動作環境に対応する SAS ドキュメント“Introduction to SAS Files, Libraries, and Engines in UNIX Environments” (*SAS Companion for UNIX Environments*)、“Introduction to SAS Files” (*SAS Companion for Windows*)、および“Using SAS Libraries” (*SAS Companion for z/OS*)を参照してください。

SAS データセット

SAS データセットには次の 2 種類があります。

- SAS データファイル
- SAS ビュー

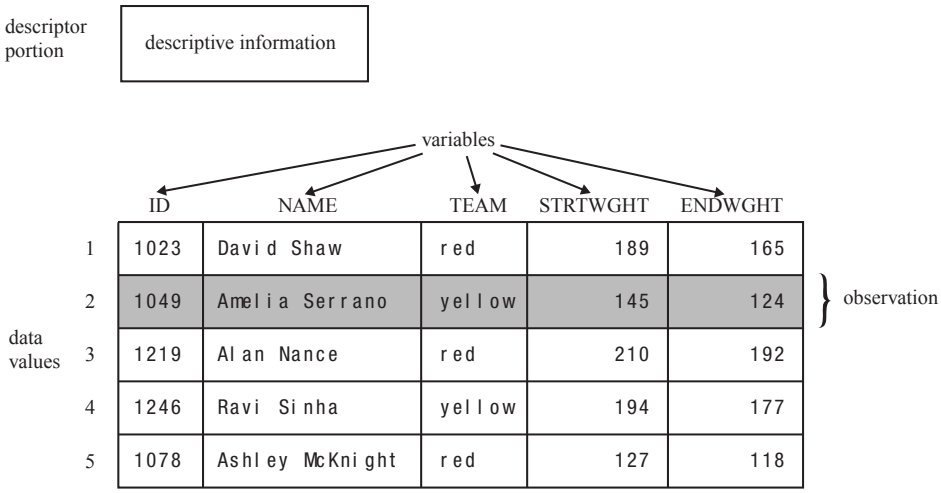
SAS データファイルは、データ値の属性を記述したディスクリプタ情報をディスクリプタ部に、データ値をデータ部に格納します。これに対して、SAS ビューは、実際には値を格納しません。SAS ビューには、データ値やディスクリプタ情報の取り出しに必要な情報だけが物理的に格納されています。SAS ビューでは、1 つまたは複数の SAS データセットに格納されているデータ、あるいは他のベンダのソフトウェアファイルに格納されているデータを参照できます。SAS ビューを使用すると、SAS データファイルで必要とされるディスク領域を使用せずに、論理的な SAS データセットを作成できます。

SAS デートセットは次のものから構成されます。

- ディスクリプタ情報
- データ値

ディスクリプタ部には、SAS データセットの属性情報が格納されています。データ部には、入力または計算されたデータ値が格納されています。これらの値は、オブザベーションと呼ばれる行、および変数と呼ばれる列の形式で編成されています。オブザベーションは、通常 1 つのオブジェクトに関連付けられるデータ値の集合です。変数は、ある所定の特性を示すデータ値の集合です。次の図は、SAS データセットを表しています。

図 1.1 SAS データセットの表現



通常、オブザベーションは、在庫品目、地域の営業所、クライアント、医療施設の患者などのエンティティに関連付けられたデータです。変数は、販売価格、在庫数、元のベンダなど、これらのエンティティの特性です。データ値が不完全な場合、SAS System は、欠損値を使用して、オブザベーション内で欠落している変数を表します。

外部ファイル

データの読み書きに使用するデータファイルのうち、SAS System によって識別されない構造を持つファイルのことを、外部ファイルと呼びます。外部ファイルを使用して、次のものを格納できます。

- SAS データファイルに読み込む生データ
- SAS プログラムステートメント
- プロシジャ出力

動作環境の情報

使用している動作環境における外部ファイルの特性の詳細については、使用している動作環境に対応する SAS ドキュメント“Using External Files and Devices” (*SAS Companion for UNIX Environments*)、“Using External Files under Windows” (*SAS Companion for Windows*)、および“Assigning External Files” (*SAS Companion for z/OS*)を参照してください。

データベース管理システムファイル

SAS ソフトウェアは、多くの一般的なデータベース管理システム(DBMS)のファイルなど、他のベンダのソフトウェアのデータを読み書きできます。Base SAS ソフトウェアに加えて、使用している DBMS および動作環境に対応した SAS/ACCESS ソフトウェアのライセンスが必要です。

SAS 言語要素

SAS 言語は、他の多くのプログラミング言語と同様、ステートメント、式、オプション、入力形式、出力形式、関数で構成されます。SAS System では、次に示す 2 つの SAS ステートメント群のいずれかでこれらを使用します。

- DATA ステップ

- PROC ステップ

DATA ステップは、次のタスクを実行する SAS 言語のステートメントのグループで構成されます。

- 外部ファイルからデータを読み込みます。
- 外部ファイルにデータを書き出します。
- SAS データセットと SAS ビューを読み込みます。
- SAS データセットと SAS ビューを作成します。

データが SAS データセットとしてアクセス可能になると、SAS プロシジャと呼ばれるツール群を使用して、そのデータを分析し、レポートを作成できます。

プロシジャステートメントは、PROC ステップと呼ばれます。SAS プロシジャは、SAS データセット内のデータを分析し、統計量、テーブル、レポート、グラフ、プロットを作成します。また、SQL クエリを作成したり、データに対して他の分析や操作を実行したりします。SAS プロシジャは、SAS ファイルの管理方法や出力方法も提供します。

DATA ステップまたは PROC ステップの外部で、SAS グローバルステートメントおよびグローバルオプションを使用することもできます。

SAS マクロ機能

Base SAS ソフトウェアには、強力なプログラミングツールである SAS マクロ機能が含まれています。マクロ機能を使用すると、SAS プログラムの拡張およびカスタマイズが可能になり、また、一般的なタスクを実行するために入力するプログラムの量を減らすことができます。マクロは、コンパイル済みマクロプログラムステートメント、および保存されたテキストを含む SAS ファイルです。マクロを使用すると、SAS ステートメントや SAS コマンドの生成、SAS ログへのメッセージの表示、入力の受け付け、マクロ変数の作成や値の変更を自動的に行うことができます。詳細については、*SAS Macro Language: Reference* を参照してください。

SAS セッションの実行方法

SAS セッションの開始

SAS セッションは、SAS コマンドを使用して開始します。SAS コマンドは、使用している動作環境の他のコマンドと同じ規則に従います。一部の動作環境では、システムコマンドまたは制御ステートメントのファイル内に SAS コマンドを記述します。また、別の動作環境では、システムプロンプトで SAS コマンドを入力する場合や、メニューから **SAS** を選択する場合があります。

SAS セッションの種類

SAS System は、次の方法のうち、お使いの動作環境で使用可能な任意の方法で実行できます。

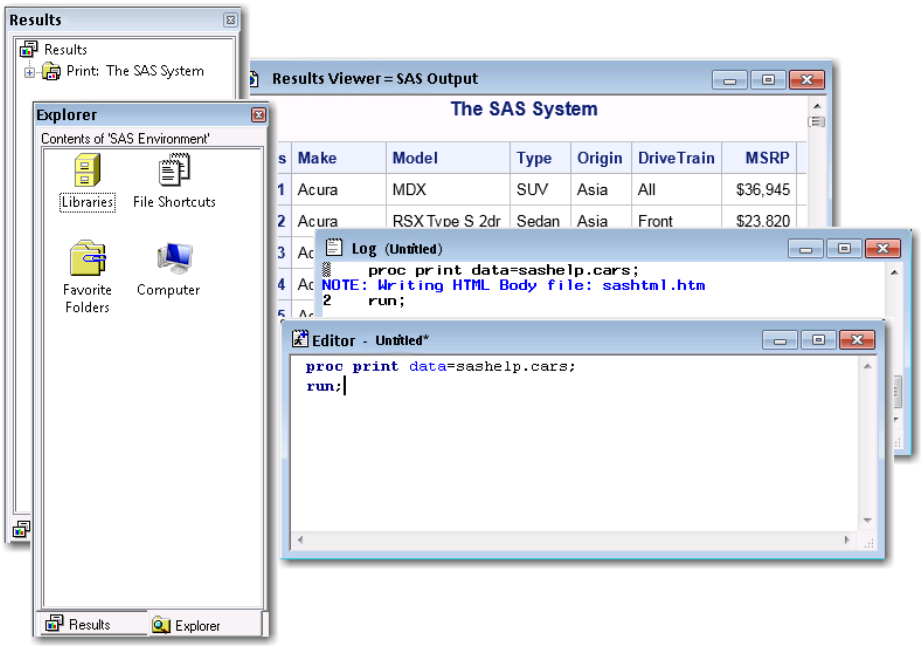
- SAS ウィンドウ環境
- 対話型ラインモード
- 非対話型モード
- バッチ(またはバックグラウンド)モード

また、SAS/ASSIST ソフトウェアは、SAS プログラムの作成および実行が可能なメニュー方式のプロダクトです。

SAS ウィンドウ環境

SAS ウィンドウ環境では、プログラムステートメントの編集および実行を行えます。また、SAS ログ、プロシジャ出力、オンラインヘルプなどを表示することもできます。次の図は、SAS ウィンドウ環境を示しています。

図 1.2 SAS ウィンドウ環境



エクスプローラウィンドウでは、ライブラリに格納されている SAS ファイルの表示と管理、および外部ファイルへのショートカットの作成ができます。**結果**ウィンドウは、サブミットした SAS プログラムからの出力の操作と管理に役立ちます。このウィンドウでは、個々の出力項目の表示、保存、管理が可能です。プログラムエディタウィンドウ、ログウィンドウ、出力ウィンドウでは、SAS プログラムの入力、編集、サブミット、SAS セッションやサブミットしたプログラムに関するメッセージの表示、サブミットしたプログラムからの出力の表示ができます。SAS ウィンドウ環境の詳細については、16 章, “SAS ウィンドウ環境の紹介” (341 ページ)を参照してください。

対話型ラインモード

対話型ラインモードでは、SAS System からの要求に応じて、順番に SAS プログラムステートメントを入力します。DATA ステップと PROC ステップは、次のいずれか 1 つ以上が起こったときに実行されます。

- データ行の後に RUN、QUIT、セミコロンだけの行が入力された場合
- 別の DATA ステートメントまたは PROC ステートメントが入力された場合
- ENDSAS ステートメントが検出された場合

デフォルトでは、プログラムステートメントの実行直後に、SAS ログと出力結果が表示されます。

非対話型モード

非対話型モードでは、SAS プログラムステートメントは外部ファイルに格納されています。ファイル内のステートメントは、そのファイルを参照する SAS コマンドを発行するとすぐに実行されます。SAS ログと出力結果は、使用している動作環境と SAS システムオプションに応じて、別の外部ファイルに書き出されるか、または表示されます。

動作環境の情報
これらのファイルの命名方法や格納場所の詳細については、使用している動作環境に対応する SAS ドキュメント"The Default Routings for the SAS Log and Procedure Output in UNIX Environments" (*SAS Companion for UNIX Environments*)、"Routing Procedure Output and the SAS Log to a File" (*SAS Companion for Windows*)、および"Destinations of SAS Output Files" (*SAS Companion for z/OS*)を参照してください。

バッチモード

バッチ実行またはバックグラウンドでの実行をサポートしている動作環境では、バッチモードで SAS ジョブを実行できます。ファイルに SAS ステートメントを記述し、サイトで必要な制御ステートメントとシステムコマンドと共にそのステートメントをサブミットして実行します。

バッチモードで SAS ジョブをサブミットすると、そのジョブの SAS ログを保存するために 1 つのファイルが作成されます。また、PROC ステップで作成された出力を保存するために、または指示がある場合は DATA ステップ内の PUT ステートメントによって作成された出力を保存するためにもう 1 つのファイルが作成されます。

動作環境の情報
バッチモードでの SAS ジョブ実行の詳細については、使用している動作環境に対応する SAS ドキュメント

- UNIX 動作環境: "Printing and Routing Output" (*SAS Companion for UNIX Environments*)
- Windows 動作環境: "Running SAS in Batch Mode" (*SAS Companion for Windows*)
- z/OS 動作環境: "Directing SAS Log and SAS Procedure Output" (*SAS Companion for z/OS*)

また、ジョブをバッチで実行したりバッチジョブからの出力を表示するためのローカルな必要条件については、サイトに固有のドキュメントを参照してください。

オブジェクトサーバーモード

オブジェクトサーバーモードの場合、SAS は IOM サーバーとして実行されます。SAS IOM サーバーの例としては、SAS Metadata Server、SAS Workspace Server、SAS Stored Process Server、SAS OLAP Server などがあります。SAS をオブジェクトサーバーモードで実行する場合の詳細については、*SAS Intelligence Platform:Application Server Administration Guide* を参照してください。

SAS セッションのカスタマイズ

デフォルトのシステムオプション設定

構成ファイル内には、必要な SAS システムオプションの設定を保存できます。SAS System の起動時に、これらのシステムオプションの設定が自動的に読み込まれて、デフォルトの設定として有効になります。SAS システムオプションによって、コンピュータハードウェアや動作環境と SAS System とのインターフェイスの初期化方法、データを読み書きする方法、出力を表示する方法、などのさまざまなグローバルな機能が決まります。

環境設定ファイル内に SAS システムオプションを設定しておくと、SAS System を起動するたびにそのオプションを指定する必要がなくなります。たとえば、環境設定ファイル内に NODATE システムオプションを指定すると、デフォルトの設定として各出力ページの上部に出力される日付が非表示になります。

動作環境の情報
環境設定ファイルの詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。一部の動作環境では、システム全体の環境設定ファイルとユーザー固有の環境設定ファイルの両方を使用できます。

ステートメントの自動実行

SAS System の起動時に特定の SAS ステートメントを自動的に実行するには、そのような SAS ステートメントを自動実行ファイル内に保存します。SAS System は、システムの起動後に自動的にそのステートメントを実行します。自動実行ファイルを有効にするには、AUTOEXEC=システムオプションを指定します。

自動実行ファイル内には、任意の SAS ステートメントを記述できます。たとえば、レポートのタイトルや脚注を出力したり、マクロやマクロ変数を作成したりする設定を行えます。

動作環境の情報
自動実行ファイルを SAS System で有効となるように設定する方法については、使用している動作環境に対応する SAS ドキュメントを参照してください。

SAS ウィンドウ環境のカスタマイズ

SAS ウィンドウ環境の多くの属性をカスタマイズできます。カスタマイズした設定を保存することで、以降のセッションでそれらの設定を使用できます。SAS ウィンドウ環境では、次のタスクを実行できます。

- **エクスプローラ**ウィンドウ内の、外観の変更と項目の表示順序の並べ替え
- メンバ、エントリ、ファイルタイプを登録することによる**エクスプローラ**ウィンドウのカスタマイズ
- お気に入りフォルダの設定
- ツールバーのカスタマイズ
- フォント、色、プリファレンスの設定

SAS ウィンドウ環境のカスタマイズ方法の詳細については、SAS オンラインヘルプを参照してください。

Base SAS の概念について

SAS System の概念

SAS System 共通の概念には、ワード(語)および SAS 名の規則、変数、欠損値、式、日付値、時間値、日付間隔値、6 つの SAS 言語要素(データセットオプション、出力形式、関数、入力形式、ステートメント、システムオプション)などのような、SAS 言語の基本要素が含まれます。

この概念には、SAS ログ、SAS 出力、エラー処理、WHERE 処理、デバッグに関する情報など、初めて SAS System を使用する場合に役立つ紹介情報もあります。SAS System の処理に関する情報は、SAS プログラムを記述するのに役立ちます。システムパフォーマンスの改善方法や、パフォーマンスの監視方法に関する情報もあります。

DATA ステップの概念

DATA ステップの本質的な概念を理解しておくと、DATA ステッププログラムを効果的に作成できます。この概念には、SAS が DATA ステップを処理する仕組み、生データを読み込んで SAS データセットを作成する方法、DATA ステップでレポートを書き出す方法などが含まれます。

さらに詳細な概念としては、SAS データセットを作成した後で情報を結合したり変更したりする方法、データの BY グループ処理を実行する方法、より効率的なプログラミングのために配列処理を使用する方法、コンパイル済みストアド DATA ステッププログラムを作成する方法などがあります。

SAS ファイルの概念

SAS ファイルの概念には、単純な SAS プログラムの作成には必須ではないものの、高度なアプリケーションに役立つトピックが含まれます。これらのトピックには、データライブラリ、データファイル、SAS ビュー、カタログ、ファイル保護、エンジン、外部ファイルなど、SAS が使用する物理ファイル構造の構成要素についての内容が含まれています。

データファイルの詳細なトピックには、監査証跡、世代データセット、整合性制約、インデックス、ファイル圧縮があります。また、旧リリースとの互換性の問題、異なる動作環境にまたがるファイルの処理方法なども含まれています。

2 章

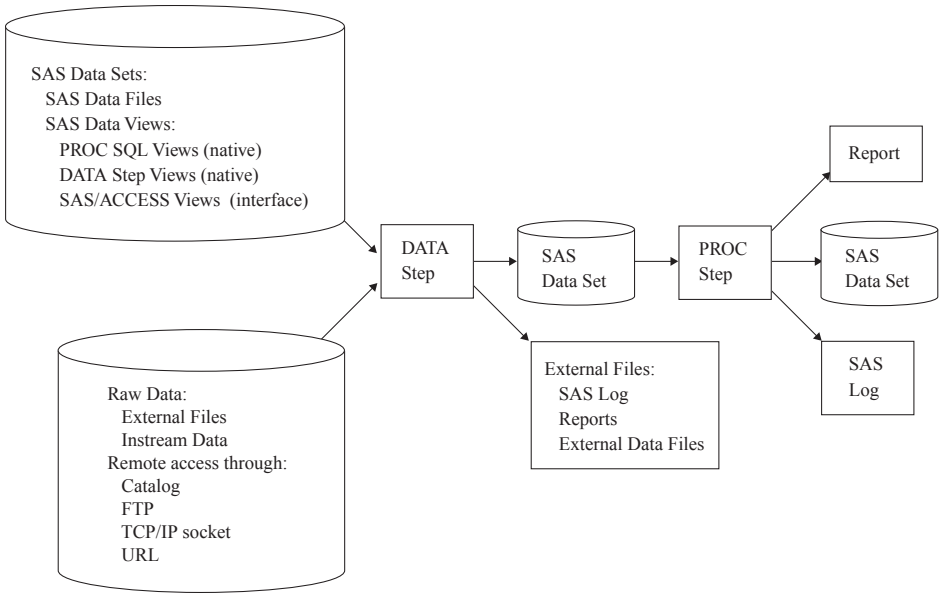
SAS の処理

SAS 処理の定義

SAS 処理とは、SAS 言語が入力データを読み込んでデータを変換し、要求された種類の出力を生成するまでの一連の方法を指します。SAS 言語を構成する 2 種類のステップ、DATA ステップと PROC (プロシジャ)ステップによって処理を行います。一般に、DATA ステップはデータを操作し、PROC ステップはデータの分析、出力の生成、SAS ファイルの管理を行います。これら 2 種類のステップは、それぞれを単独で使用することも組み合わせて使用することもでき、SAS プログラムの基盤を構成しています。

次の図は、DATA ステップと PROC ステップを使用する SAS System の処理の概要を示しています。この図は、主として DATA ステップについて記述しています。

図 2.1　SAS の処理



DATA ステップへの入力ソースとして各種のデータを使用できます。DATA ステップには、データを処理する命令を含む SAS ステートメントが含まれています。SAS プログラム内の各 DATA ステップのコンパイル時または実行時に、SAS System は処理メッセージとエラーメッセージを含む SAS ログを生成して表示します。これらのメッセージは、SAS プログラムのデバッグ時に役立つ情報を提供します。

SAS プログラムの入力の種類

SAS プログラムでは、次のさまざまな入力データソースからデータを読み込むことができます。

SAS データセット
　次の 2 つの種類があります。

SAS データファイル	実際のデータ値が含まれています。SAS データファイルは、ファイル内のデータ値の属性を記述したディスクリプタ部と、データ部で構成されています。
SAS ビュー	他の場所に格納されているデータへの参照が含まれます。SAS ビューは、別の場所に格納されたデータを参照するデータセットで、ディスクリプタ情報やデータ値の取り出しに必要な情報だけを物理的に格納しています。SAS ビューにより、ディスク領域を節約しながら、各種のソースから動的に結合した新しいデータセットを作成することができます。SAS ビューには、DATA ステップビュー、PROC SQL ビュー、SAS/ACCESS ビューがあります。ほとんどの場合、SAS ビューは、SAS データファイルを扱うのと同様に使用できます。

詳細については、26 章, “SAS データファイル” (597 ページ) および 27 章, “SAS ビュー” (665 ページ)を参照してください。

生データ
　SAS データセットに読み込まれていない、処理される前のデータです。生データは、次の 2 つのデータソースから読み込むことができます。

外部ファイル　フォーマットされたデータ(列に編成されたデータ)またはフリーフォーマットのデータ(列に編成されていないデータ)で構成されたレコードが記述されたファイルです。

入力ストリームデータ　SAS プログラムの内部に記述されるデータです。データの先頭に DATALINES ステートメントを使用して、入力ストリームデータを指定します。

生データの詳細については、19 章, “生データの読み込み” (417 ページ)を参照してください。

リモートアクセス

TCP/IP ソケットや URL など、従来型ではないソースから入力データを読み込むことを可能にします。SAS System は、外部ファイルのデータを扱うのと同様に、リモートアクセスのデータを扱います。入力データにリモートからアクセスするには、次の方式があります。

SAS カタログ　このアクセス方式を指定すると、SAS カタログを外部ファイルとして参照できます。

クリップボード　このアクセス方式を指定すると、ホストコンピュータ上のクリップボードに対してテキストデータを読み書きできます。

DATAURL　このアクセス方式を指定すると、DATAURL 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。

FTP　ファイル転送プロトコル(FTP)によるアクセス方式です。ネットワークに接続されており FTP サーバーが稼働しているホストコンピュータからファイルの読み書きを実行できるようにします。

Hadoop　このアクセス方式を指定すると、Hadoop 分散ファイルシステム(HDFS)上のファイルにアクセスできるようになります。HDFS のロケーションは構成ファイルで指定します。

SFTP　セキュアファイル転送プロトコル(SFTP)によるアクセス方式です。ネットワークに接続されており Open SSH SSHD サーバーが稼働しているホストコンピュータからファイルの読み書きを実行できるようにします。

TCP/IP ソケット　このアクセス方式を指定すると、伝送制御プロトコル/インターネットプロトコル(TCP/IP)ソケットからの読み込みや TCP/IP ソケットへの書き込みを実行できます。

URL　URL (Uniform Resource Locator)によるアクセス方式です。ネットワークに接続されていて、URL サーバーが稼働しているホストコンピュータからファイルの読み書きを実行できるようにします。

WebDAV　WebDAV プロトコルによるアクセス方式です。ネットワークに接続されていて、WebDAV サーバーが稼働しているホストコンピュータからファイルの読み書きを実行できるようにします。

ZIP　zlib サービスを使用して ZIP ファイルにアクセスできるようにします。

データのリモートアクセスの詳細については、次のトピックを参照してください。

- “FILENAME Statement, CLIPBOARD Access Method” (*SAS Statements: Reference*)
- “FILENAME Statement, CATALOG Access Method” (*SAS Statements: Reference*)

- “FILENAME Statement, DATAURL Access Method” (*SAS Statements: Reference*)
- “FILENAME Statement, FTP Access Method” (*SAS Statements: Reference*)
- “FILENAME Statement, Hadoop Access Method” (*SAS Statements: Reference*)
- “FILENAME Statement, SFTP Access Method” (*SAS Statements: Reference*)
- “FILENAME Statement, SOCKET Access Method” (*SAS Statements: Reference*)
- “FILENAME Statement, URL Access Method” (*SAS Statements: Reference*)
- “FILENAME Statement, WebDAV Access Method” (*SAS Statements: Reference*)
- “FILENAME Statement, ZIP Access Method” (*SAS Statements: Reference*)

DATA ステップ

DATA ステップで可能な処理について

DATA ステップは入力データを処理します。また、DATA ステップでは、SAS データセットが作成されます。作成される SAS データセットは、SAS データファイルまたは SAS ビューのどちらかです。DATA ステップへの入力データには、生データ、リモートアクセス、割り当てステートメント、SAS データセットを使用します。DATA ステップでは、値の計算、処理対象の特定の入力レコードの選択、条件付きロジックの使用などができます。DATA ステップからの出力にはいくつかの種類があります。たとえば、SAS データセットやレポートを出力します。また、SAS ログまたは外部データファイルにデータを出力することもできます。詳細については、18 章, “DATA ステップの処理” (389 ページ)を参照してください。

DATA ステップの出力

DATA ステップからは、作成した SAS データセットを出力できます。プログラムのログ、レポート、外部データファイルなどの外部ファイルにも、DATA ステップから出力できます。また、新しいデータセットを作成しないで、既存のデータセットを更新することも可能です。多くの SAS プロシジャでは、処理対象のデータは SAS データセットの形式であることが必要なので、DATA ステップによりさまざまなデータ操作を行います。DATA ステップからは、次に示す出力を作成できます。

SAS ログ
: 処理メッセージとプログラムエラーが表示されます。SAS ログは、デフォルトで作成されます。

SAS データファイル
: データ部とデータ値の属性を記述したデータディスクリプタ部を含む SAS データセットです。

SAS ビュー
: SAS ビューとは、別のファイルに格納されているデータ部とディスクリプタ情報の取り出しに必要な情報だけを物理的に格納する SAS データセットです。SAS ビューを使用することにより、ディスク領域を使用して新しいデータセットを作成することなく、さまざまな入力データのソースから取得したデータを動的に結合したデータセットを作成することができます。SAS データファイルには、実際のデータ値が含まれています。一方、SAS ビューには、他の場所に格納されているデータへの参照のみが含まれます。SAS ビューは、メンバタイプ VIEW を持つ SAS ファイルとして管

理されます。ほとんどの場合、SAS ビューは、SAS データファイルを扱うのと同様に使用できます。

外部データファイル
DATA ステップの処理の結果が含まれます。このファイルは、データファイルまたはテキストファイルです。データは、フォーマットされたレコード、またはフリーフォーマットのレコードです。

レポート
DATA ステップの処理の結果が含まれます。通常は、PROC ステップを使用してレポートを生成しますが、次の 2 つの種類のレポートは DATA ステップから生成できます。

プロシジャ出力ファイル	DATA ステップの処理の出力結果が含まれます。通常はヘッダーと改ページを含みます。
HTML ファイル	World Wide Web で表示できる結果が含まれます。この種類の出力は、ODS (アウトプットデリバリシステム)を経由して生成されます。

PROC ステップ

PROC ステップで可能な処理について

PROC ステップは、プロシジャを呼び出して実行する一連の SAS ステートメントで構成されます。通常は、SAS データセットを入力として使用します。PROC ステップを使用して、SAS データセット内のデータの分析、フォーマットされたレポート生成、処理結果の出力をします。また、PROC ステップは SAS ファイルの管理方法を提供します。PROC ステップを使用して、入力データから必要な出力を容易に生成できます。また、PROC ステップでは関数を実行でき、SAS データセットに関する情報の表示も行えます。SAS プロシジャの詳細については、*Base SAS Procedures Guide* を参照してください。

PROC ステップの出力

PROC ステップから提供される出力は、単変量記述統計、度数表、クロス集計表、単変量記述統計で構成される表形式レポート、チャート、プロットなどがあります。また、更新されたデータセットを出力する場合もあります。プロシジャ出力の詳細については、*Base SAS Procedures Guide* および *SAS Output Delivery System: User's Guide* を参照してください。

サーバーがロックダウン状態になった場合の SAS 処理の制限

概要情報

SAS をクライアント/サーバー環境で実行している(たとえば、SAS Enterprise Guide を使用している)場合、SAS サーバー管理者は、ホストシステム上にあるファイルやディレクトリに対するアクセスを制限できます。また、SAS セッションがロックダウン状態になると、特定のアクセス方式、関数、CALL ルーチン、プロシジャがデフォルトで制限さ

れます。詳細については、次の文書を参照してください。“Locked-Down SAS Sessions” (*SAS/CONNECT User's Guide*)

SAS がロックダウン状態になると、次に示す SAS 言語要素は使用できなくなります。

関数と CALL ルーチン	アクセス方式	プロシジャ	その他
ADDR 関数	EMAIL	GROOVY プロシジャ	DATA ステップ Java オブジェクト
ADDRLONG 関数	FTP	HADOOP プロシジャ	
CALL MODULE	HADOOP	HTTP プロシジャ	
CALL POKE ルーチン	HTTP	JAVAINFO プロシジャ	
CALL POKELONG ルーチン	SOCKET	SOAP プロシジャ	
PEEK 関数	TCPIP		
PEEKC 関数	URL		
PEEKCLONG 関数			
PEEKLONG 関数			

LOCKDOWN ステートメントの ENABLE_AMS=オプションを使うと、管理者は、LOCKDOWN が有効である場合に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。LOCKDOWN ステートメントの ENABLE_AMS=オプションを使うと、次のアクセス方式やプロシジャを再有効化できます。

ENABLE_AMS=オプション値
FTP
EMAIL
HADOOP (PROC HADOOP を有効化)
HTTP (PROC HTTP と PROC SOAP を有効化)
SOCKET
TCPIP
URL (PROC HTTP と PROC SOAP を有効化)

ロックダウンされているリソースの使用を試みると、SAS セッションに SAS ログ機能が構成されている場合、SAS は、エラーメッセージを SAS ログに出力するか、または **Audit.Lockdown** ロガーに出力します。

詳細については、次の文書を参照してください。

- SAS Intelligence Platform Documentation ページ(support.sas.com/documentation/onlinedoc/intellplatform)の *SAS Intelligence Platform: Application Server Administration Guide* の“LOCKDOWN system option”および“LOCKDOWN statement”。
- SAS Intelligence Platform Documentation ページ(support.sas.com/documentation/onlinedoc/intellplatform)の *SAS Intelligence Platform: Security Administration Guide* の“Locked-Down Servers”。

- LOCKDOWN をサポートしていない製品のリストについては"TS2m1 products and solutions that do not support the lockdown feature" (SAS Usage Note 51644)を参照してください。

z/OS 固有の情報

制限される機能

永久 z/OS データセットならびに UFS ファイルおよびディレクトリへのアクセスは、ロックダウンリストで有効化されない限り許可されません。この制限はすべての SAS 機能(特に、サーバーでの実行のためにサブミットされる SAS プログラムの FILENAME および LIBNAME ステートメント)に適用されます。また、SAS Enterprise Guide など、SAS クライアント経由でサーバー上のファイルをリストする機能にも適用されます。SAS がロックダウン状態の場合、サーバー管理者によって設定された外部割り当てのデータ定義名によるアクセスを除いて、アンカタログされた z/OS データセットへのアクセスは許可されません。ただし、一時 z/OS データセットおよび UFS ファイルの作成、ならびにシングルクライアントセッションにおけるそのデータセットやファイルの処理に対する制限はありません。z/OS データセットは、DISP=(NEW,DELETE)を割り当てられている場合、一時的と見なされます。外部ファイルは、TEMP の FILENAME デバイスを使用して割り当てられている場合、一時的と見なされます。クライアント WORK ライブラリのメンバはすべて一時的と見なされます。

インストール時の SAS サーバー管理者が、ロックダウンリストの内容についての責任者です。したがって、ロックダウン状態で使用不可能な z/OS データセットまたは UFS ファイルへのアクセスが必要な場合は、サーバー管理者に連絡してください。

無効化される機能

次の SAS プロシジャは、z/OS に固有であり、SAS がロックダウン状態の場合は実行できません。

PDS	SOURCE
PDSCOPY	TAPECOPY
RELEASE	TAPELABEL

次の DATA ステップ関数は、z/OS に固有であり、SAS がロックダウン状態の場合は実行できません。

ZVOLLIST	ZDSATTR
ZDSLIST	ZDSRATT
ZDSNUM	ZDSXATT
ZDSIDNM	ZDSYATT

次のアクセス方式は、z/OS に固有であり、SAS がロックダウン状態の場合は実行できません。

VTOC

ロックダウンパスリストでの関数の指定

関数を指定中の SAS セッションがロックダウン状態で、その関数で指定されているパス名がロックダウンパスリストに追加されていない場合、その関数は失敗し、そのロックダウンデータに関連するファイルアクセスエラーは、SYSMSG 関数を指定しない限り、SAS ログに生成されません。

DATA ステップで関数呼び出しの後に SYSMSG 関数を入れると、ロックダウン関連のファイルアクセスエラーを表示させられます。

次の関数ばかりでなく、物理的なパス名の場所を入力としているその他の関数についても、この条件が真になります。

- DCREATE
- FILEEXIST
- FILENAME
- RENAME
- DSNCATLGD (z/OS-specific)

3 章

SAS 言語のワードと命名規則について

SAS 言語におけるワード

ワードの定義

SAS 言語のワードまたはトークンは、SAS System に指示を伝達する文字の集合であり、SAS プログラムにおける最小単位となるものです。1 つのワードには最大で 32,767 文字を格納できます。

SAS System は、次のいずれかを検出すると、ワードまたはトークンの境界として認識します。

- 新しいトークンの先頭
- 名前トークンまたは数値トークンの後のブランク
- リテラルトークンの末尾の引用符

SAS 言語のワードまたはトークンは、次の 4 種類のいずれかに分類されます。

- 名前
- リテラル
- 数値
- 特殊文字

ワードまたはトークンの種類

ワードまたはトークンには、次の 4 種類があります。

名前
アルファベットまたはアンダースコア(_)で始まる一連の文字です。後続の文字には、アルファベット、アンダースコア(_)、数字を使用できます。名前トークンは最大で 32,767 文字を格納できます。ただし、ほとんどの場合、SAS 名の最大長はこれより短い 32 文字や 8 文字までに制限されています。表 3.1 (25 ページ)を参照してください。名前トークンの例を次に示します。

- `data`
- `_new`
- `yearcutoff`
- `year_99`
- `descending`
- `_n_`

リテラル
一重引用符(')または二重引用符(")で囲まれた 1 - 32,767 個の文字列で構成されます。リテラルの例を次に示します。

- `'Chicago'`
- `"1990-91"`
- `'Amelia Earhart'`
- `'Amelia Earhart''s plane'`
- `"Report for the Third Quarter"`

注: トークンを囲む引用符は、そのトークンがリテラルであることを示します。これらの引用符はリテラルトークンの一部としては保存されません。

数字
通常は、数字で構成されます。小数点や先頭に正符号(+)または負符号(-)が付く場合があります。SAS System は、指数表記(E−表記)、16 進表記、欠損値記号、SAS 日付値や時間値などの形式の数値も数値トークンとして認識します。数値トークンの例を次に示します。

- `5683`
- `2.35`
- `0b0x`
- `-5`
- `5.4E-1`
- `'24aug90'd`

特殊文字
通常は、アルファベット、数字、アンダースコア(_)、ブランク以外の文字です。一般的には、1 つの特殊文字が 1 つのトークンになります。ただし、**や<=など、2 文字の演算子が 1 つのトークンになる場合もあります。ブランクは、名前トークンまたは数値トークンの終わりを示しますが、ブランク自体はトークンではありません。特殊文字トークンの例を次に示します。

- =
- ;
- '
- +
- @
- /

SAS ステートメント内におけるワードの配置とワード間のブランク挿入

ワード間のブランク挿入に関する規則

SAS ステートメント内におけるワード間のブランク挿入に関する規則を次に示します。

- SAS ステートメントは行の任意の列位置から開始でき、同じ行に複数のステートメントを記述できます。
- ステートメントは複数の行にまたがって記述できますが、1 つのワードを 2 行に分けることはできません。
- SAS ステートメント内のブランクは文字として処理されません。ただし、引用符に囲まれているブランクは、リテラルまたはリテラルの一部として扱われます。したがって、SAS ステートメントで、ブランクを 1 つ挿入できる場所に複数のブランクを挿入できます。ブランクを挿入しても、構文には影響しません。
- SAS プログラムでのワード(トークン)間のブランクの入れ方は、ワード(トークン)の境界を認識する規則によって決まります。SAS System が演算子などの記号によってトークンの先頭を特定できる場合は、ブランクを入れる必要はありません。SAS System が各トークンの先頭を特定できない場合は、ブランクを入れる必要があります。例 (23 ページ)を参照してください。

SAS System のブランクに関する規則は厳しくありませんが、ステートメントのインデント方法を一貫させておくと、SAS プログラムが読みやすく、保守しやすくなります。ブランクについて役立つ規則の例を次に示します。

Examples

- 次に示すステートメントでは、SAS System は次のトークンの先頭を調べて、各トークンの境界を特定できるため、ブランクは必要ありません。

```
total=x+y;
```

最初の特殊文字トークンである等号(=)は、名前トークン **total** の終わりを示します。次の特殊文字トークンである正符号(+)は、名前トークン **x** の終わりを示します。最後の特殊文字トークンであるセミコロン(;)は、**y** トークンの終わりを示します。この例では、トークンの終わりにブランクは必要ありません。しかし、次のようにブランクを追加することで読みやすくなります。

```
total = x + y;
```

- 次のステートメントでは、ブランクが必要です。これは、ブランクがないと SAS System が各トークンを認識できないためです。

```
input group 15 room 20;
```

ブランクを挿入しないと、セミコロンまでのステートメント全体が、名前トークンとして扱われます。名前トークンは、アルファベットまたはアンダースコア(_)で始まり、それ以降にアルファベット、数字、アンダースコア(_)が続き、長さは 32,767 文字以下

です。したがって、このステートメントには、各名前トークンと数字トークンを区切るブランクが必要です。

SAS 言語における命名規則

SAS 名の定義

SAS 名とは、次のような SAS 言語要素を表す名前トークンです。

- 変数
- データセット
- 出力形式
- 入力形式
- プロシジャ
- オプション
- ステートメントラベル
- ライブラリ参照名またはファイル参照名
- カタログエントリ
- 配列
- SAS マクロまたはマクロ変数
- コンポーネントオブジェクト

SAS 名には、次の 2 種類があります。

- SAS 言語要素の名前(システムが提供する名前)
- ユーザー指定の名前

次のセクションでは、ユーザー指定の名前について説明します。

ユーザー指定の SAS 名の規則

大部分の SAS 名の規則

次のリストに、SAS 名を作成する際に使用する規則を示します。

注: 規則は、SAS 変数名、データセット名、ビュー名、アイテムストア名の場合、他の言語要素よりも柔軟です。“SAS 変数名の規則” (26 ページ)および“SAS データセット名、ビュー名、アイテムストア名の規則” (27 ページ)を参照してください。

- SAS 名の長さは、その SAS 名が割り当てられる言語要素によって異なります。SAS 名の最大の長さは多くの場合 32 文字ですが、最大の長さが 8 文字の場合もあります。SAS 名とその最大長のリストについては、表 3.1 (25 ページ)を参照してください。
- 最初の文字は、アルファベット(A, B, C, .., Z)またはアンダースコア(_)でなければなりません。以降の文字には、アルファベット、数字(0, 1, ..., 9)、またはアンダースコア(_)を使用できます。
- アルファベットの大文字と小文字を混在させることができます。
- SAS 名の中にブランクは使用できません。
- アンダースコア(_)以外の特殊文字は使用できません。ファイル参照名でのみ、ドル記号($)、シャープ記号(#)、アットマーク(@)を使用できます。
- SAS System では、自動変数、変数リスト、SAS データセット、ライブラリ参照名のために、いくつかの名前が予約されています。
  - 変数を作成する場合は、特殊な SAS 自動変数の名前(_N_や_ERROR_など)と特殊な変数リストの名前(_CHARACTER_、_NUMERIC_、_ALL_など)は使用しないでください。

  - SAS ライブラリにライブラリ参照名を割り当てる場合、次のライブラリ参照名を名前に使用しないでください。
    - Sashelp
    - Sasmsg
    - Sasuser
    - Work
  - SAS データセットを作成する場合、次の名前を使用しないでください。
    - _NULL_
    - _DATA_
    - _LAST_
- 外部ファイルにファイル参照名を割り当てる場合、ファイル名 SASCAT を使用しないでください。
- マクロ変数を作成する場合、SYS で始まる名前を使用しないでください。

表 3.1 *ユーザー指定の SAS 名の最大文字数*

ユーザー指定の SAS 名	最大文字数
配列	32
CALL ルーチン	16
カタログエントリ	32
コンポーネントオブジェクト	32
DATA ステップのステートメントラベル	32
DATA ステップの変数ラベル	256
DATA ステップの変数名	32
DATA ステップウィンドウ	32
エンジン	8
ファイル参照名	8
出力形式、文字	31
出力形式、数字	32
関数	16
世代データセット	28
入力形式、文字	30

ユーザー指定の SAS 名	最大文字数
入力形式、数字	31
ライブラリ参照名	8
マクロ変数	32
マクロウィンドウ	32
マクロ	32
世代データセットを除く、SAS ライブラリのメンバ(SAS データセット、SAS ビュー、カタログ、インデックス)	32
パスワード	8
プロシジャ名(最初の 8 文字は一意にする必要があります。“SAS”で始めることはできません)	16
SCL 変数	32

SAS 変数名の規則

さらに多くの機能を提供するために、SAS 変数名の規則が拡張されました。VALIDVARNAME=システムオプションの設定によって、SAS セッションで作成できる変数に適用される規則、および既存のデータセットから読み込む変数に適用される規則を変更することができます。

VALIDVARNAME=システムオプションには、3 つの設定(V7、UPCASE、ANY)があります。各設定は、次に示すように、変数名の柔軟性の程度が異なります。SAS セッションで VALIDVARNAME=システムオプションを省略した場合、デフォルト値である V7 が現在の SAS セッションに自動的に割り当てられます。VALIDVARNAME=システムオプションを使用した場合の変数名に関する規則を次の表に示します。

V7

デフォルト設定です。

変数名は次の規則に従います。

- SAS 変数名の最大長は 32 文字です。
- 最初の文字は、アルファベット(A–Z、a–z)またはアンダースコアで始まる必要があります。以降の文字は、文字、アルファベット、数字、アンダースコアのいずれでもかまいません。
- 後置ブランクは無視されます。変数名は左揃えで配置されます。
- 変数名には、ブランクまたは特殊文字(アンダースコアを除く)を使用できません。
- 変数名には、大文字と小文字を混在して使用できます。SAS は、変数への最初の参照で使用された大文字と小文字の組み合わせと同じものを使用して、その変数を格納および出力します。ただし、SAS が変数名を処理する場合、SAS は内部で小文字を大文字に変換します。したがって、大文字と小文字の組み合わせだけが異なる同じ変数名を使用して、異なる変数を表現することはできません。たとえば、`cat`、`Cat`、`CAT` はすべて同じ変数を表します。

- 特殊な SAS 自動変数の名前(_N_や_ERROR_など)や変数リストの名前(_NUMERIC_、_CHARACTER_、_ALL_など)を変数に割り当てないでください。

例 season='summer';

percent_of_profit=percent;

UPCASE
旧バージョンの SAS System のように、変数名が大文字であることを除いて V7 と同じです。

ANY

- 名前は、ブランク、各国固有の文字、特殊文字、マルチバイト文字を含む任意の文字で始めることができます。
- 名前の最大長は 32 バイトです。
- 名前は Null バイトを格納できません。
- 前置ブランクは保持されますが、後置ブランクは無視されます。
- 名前には少なくとも 1 つの文字が含まれている必要があります。すべてがブランクの名前を指定することはできません。
- 名前には大文字と小文字を混在させることができます。SAS は、変数への最初の参照で使用された大文字と小文字の組み合わせと同じものを使用して、その変数を格納および出力します。ただし、SAS が変数名を処理する場合、SAS は内部で小文字を大文字に変換します。したがって、大文字と小文字の組み合わせだけが異なる同じ変数名を使用して、異なる変数を表現することはできません。たとえば、`cat`、`Cat`、`CAT` はすべて同じ変数を表します。

要件 VALIDVARNAME=システムオプションが V7 に設定されていて有効な文字(アルファベット、数字、アンダースコア)以外の文字を使用する場合は、変数名を名前リテラルとして表し、VALIDVARNAME=ANY を設定する必要があります。名前にパーセント記号(%)かアンパサンド(&)のどちらかを含む場合は、SAS マクロ機能との対話処理を避けるために名前リテラルに一重引用符を使用する必要があります。"SAS 名前リテラル" (31 ページ)および"名前リテラルの使用時のエラーの回避" (33 ページ)を参照してください。

参照項目 "SAS 名の長さをバイト数で測定すると何文字使用できるか" (30 ページ)

例 '% of profit'n=percent;

'items@warehouse'n=itemnum;

注意 **SAS 全体で、名前リテラル構文に 32 バイト制限を超える変数名を指定したり、埋め込まれている引用符が多すぎたりする場合、予期しない結果になる可能性があります。**VALIDVARNAME=ANY システムオプションの目的は、埋め込みブランクや各国固有文字などを使用可能にすることなど、他の DBMS 変数(列)命名規則との互換性を確保することにあります。

SAS データセット名、ビュー名、アイテムストア名の規則

3 種類の SAS メンバ、SAS データセット、ビュー、アイテムストアの機能が拡張されています。VALIDMEMNAME=システムオプションは、SAS セッションのこのメンバの名前にどのルールが適用されるかを指定します。VALIDMEMNAME=オプションには 2

つの設定(COMPATIBLE および EXTEND)があります。どちらも、データセット名、ビュー名、アイテムストア名のさまざまな柔軟性を備えています。

COMPATIBLE
SAS データセット名、ビュー名、アイテムストア名が次の規則に従う必要があることを示します。

- 名前の最大長は 32 文字です。
- 名前は、アルファベット(A–Z、a–z)またはアンダースコアで始まる必要があります。以降の文字は、文字、アルファベット、数字、アンダースコアのいずれでもかまいません。
- 名前には、ブランクまたは特殊文字(アンダースコアを除く)を使用できません。
- 名前には大文字と小文字を混在させることができます。SAS は内部的にメンバ名を大文字に変換します。したがって、大文字と小文字の組み合わせだけが異なる同じメンバ名を使用して、異なる変数を表現することはできません。たとえば、**customer**、**Customer**、**CUSTOMER** はすべて同じメンバ名を表します。ディスク上の名前がどのように表示されるかは、動作環境によって決まります。

別名 COMPAT

EXTEND
SAS データセット名、SAS ビュー名、アイテムストア名が次の規則に従う必要があることを示します。

- 名前には各国固有の文字を使用できます。
- 名前には特殊文字を使用できます(ただし、/ \ * ?" < > |:-は除く)。

  注: SPD Engine では、‘.’(ピリオド)をメンバ名に使用できません。
- 名前には少なくとも 1 つの文字(アルファベット、数字、有効な特殊文字、各国文字)が含まれている必要があります。
- 名前の最大長は 32 バイトです。
- Null バイトを使用できません。
- 名前はブランクまたは‘.’(ピリオド)で始めることはできません。

  注: SPD Engine には、メンバ名の先頭に‘$’を使用できません。
- 前置ブランクと後置ブランクは、メンバの作成時に削除されます。
- 名前には大文字と小文字を混在させることができます。SAS は内部的にメンバ名を大文字に変換します。したがって、大文字と小文字の組み合わせだけが異なる同じメンバ名を使用して、異なる変数を表現することはできません。たとえば、**customer**、**Customer**、**CUSTOMER** はすべて同じメンバ名を表します。名前がどのように表示されるかは、動作環境によって決まります。

制限事項 VALIDMEMNAME の値にかかわらず、メンバ名の末尾には、特殊文字#に続く 3 桁の数字を付けることはできません。このようなメンバ名は、世代データセットの命名規則と衝突するためです。このようなメンバ名を使用するとエラーになります。

VALIDMEMNAME=EXTEND が設定されている場合、ウィンドウ環境は Program、Log、Output ウィンドウの拡張規則をサポートします。ほとんどの SAS ウィンドウでは、これらの拡張規則はサポートされません。たとえば、これらの規則は SAS エクスプローラ、**VIEWTABLE** ウィンドウ、Solutions メニューで開いたウィンドウではサポートされません。

要件 VALIDMEMNAME=EXTEND の場合、SAS データセット名、SAS データビュー名およびアイテムストア名は、名前にブランク、特殊文字または各国文字が含まれているときは、SAS 名前リテラルとして書き出される必要があります。パーセント記号(%)かアンパサンド(&)のどちらかを使用する場合は、SAS マクロ機能との対話処理を避けるために名前リテラルに一重引用符を使用する必要があります。詳細については、“SAS 名前リテラル” (31 ページ)を参照してください。

動作環境 Windows および UNIX 動作環境では、VALIDMEMNAME=EXTEND が設定されている場合、すべての Base SAS ウィンドウで拡張規則がサポートされます。

Windows および UNIX 動作環境では、SAS ファイルを物理名によって直接参照する場合、最後の埋め込みピリオドが拡張子の区切り文字になります。.物理ファイル参照名にピリオドのある SAS メンバ名が含まれている場合、ファイル拡張子を追加する必要があります。たとえば、データセット名 my.member を物理ファイルとして参照する場合は、SET ステートメント set './saslib/my.member.sas7bdat'のように、参照名にファイル拡張子 sas7bdat を追加します。

z/OS 固有 VALIDMEMNAME=EXTEND が設定されている場合、Base SAS ウィンドウ環境はエディタ、ログ、アウトプットウィンドウで拡張規則をサポートします。その他の SAS ウィンドウ(VIEWTABLE ウィンドウなど)では拡張規則をサポートしません。

SAS ファイルを物理名によって直接参照する場合、ピリオドの後に有効な SAS 拡張子が続いているときに限り、最後の埋め込みピリオドが拡張子区切り文字であると見なされます。それ以外の場合は、ピリオドがメンバ名の一部であると見なされます。たとえば、名前 my.member では、member はファイル拡張子ではなくメンバ名の一部と見なされます。名前 "my.member.sas7bdat"では、メンバ名は"my.member"、ファイル拡張子は sas7bdat です。

ヒント 名前は大文字で表示されます。

参照項目 “SAS 名の長さをバイト数で測定すると何文字使用できるか” (30 ページ)

例

```
data "August Purchases"n;
```

```
data ‘Años de empleo’n.;
```

注意 **SAS 全体で、名前リテラル構文に 32 バイト制限を超える SAS メンバ名を指定したり、埋め込まれている引用符が多すぎたりする場合、予期しない結果になる可能性があります。** VALIDMEMNAME=EXTEND システムオプションの目的は、埋め込みブランクや各国固有文字などを使用可能にすることなど、他の DBMS メンバ命名規則との互換性を確保することにあります。

注: VALIDMEMNAME=オプションは、テープエンジン V9TAPE、V8TAPE、V7TAPE、V6TAPE では無効です。

SAS 名の長さをバイト数で測定すると何文字使用できるか

VALIDVARNAME=ANY または VALIDMEMNAME=EXTEND の場合、次の SAS 名の長さをバイト数で測定する必要があります。

システムオプション設定	バイトで測定した SAS 名	最大バイト数
VALIDVARNAME=ANY	変数名	32
VALIDMEMNAME=EXTEND	SAS データセット名 SAS ビュー名 アイテムストア名	32

これらのシステムオプション値が設定された場合、SAS 変数名、データセット名、ビュー名、アイテムストア名に使用できる最大文字数は、1 つの文字を格納するためのバイト数で決まります。この値は、SAS セッションの SAS エンコーディング値で設定されます。各国語サポート(NLS)文字を使用できるように VALIDVARNAME=ANY または VALIDMEMNAME=EXTEND を設定する必要があります。それ以外の場合は、1 バイト文字のみを使用できます。

西洋言語の SAS エンコーディングは、1 文字を格納するために 1 バイトのストレージを使用します。そのため、西洋言語ではこの SAS 名で 32 文字を使用できます。アジア言語の SAS エンコーディングは、1 - 2 バイトのストレージを使用して 1 文字を格納します。Unicode エンコーディング(UTF-8)は、1 つの文字の 1 - 4 バイトのストレージをサポートします。SAS エンコーディングは、1 文字を格納するために 4 バイトを使用すると、いずれかの SAS 名の最大長は 8 文字です。

SAS エンコーディングは、1 バイト文字として A - Z および a - z をサポートします。

SAS 名に使用できる最大文字数を確認するには、次の手順に従ってください。

1. SAS エンコーディングを次のいずれかの方法で見つけます。
   - ENCODING=システムオプションを、**SAS System Options** ウィンドウで検索します。
     1. コマンドバーに **options** と入力します。
     2. **Options** を右クリックし、**Find Option** を選択します。
     3. **encoding** と入力して **OK** をクリックします。
   - エディタウィンドウで、OPTIONS プロシジャで ENCODING=システムオプションを指定します。

     ```
     proc options option=encoding;
     run;
     ```
2. 表“SBCS, DBCS, and Unicode Encoding Values Used to Transcode Data”で、SAS エンコーディングでの 1 文字当たりの最大バイト数を検索します。この表は、*SAS National Language Support (NLS): Reference Guide* にあります。
3. SAS 名の最大バイト数を表 3.1 (25 ページ)から見つけます。この数字を 1 文字当たりのバイト数で割ります。結果が、SAS 名で使用可能な最大文字数です。

SAS 名前リテラル

SAS 名前リテラルの定義

SAS *名前リテラル*とは、ユーザー指定の名前トークンであり、引用符で囲まれた文字列とその後に続く大文字または小文字の *n* で表されます。ほとんどの SAS 名で使用できる文字は、_、A–Z、および a–z のみです。名前リテラルを用いると、本来は使用できない文字(ブランクや各国固有の文字など)を使用できるようになります。

3 種類の SAS 名で名前リテラルを使用できます。

- DBMS 列名
- DBMS テーブル
- アイテムストア
- SAS データセット
- SAS ビュー
- ステートメントラベル
- 変数

_、A - Z、または a - z 以外の名前リテラルで文字を使用するには、VALIDVARNAME=ANY または VALIDMEMNAME=EXTEND システムオプションを設定する必要があります。次の表に、SAS 名前リテラルを使用するために設定する必要があるオプションを示します。

表 3.2 SAS 名前リテラルシステムオプション要件

SAS 名の種類	名前リテラルの要件
DBMS 列	VALIDVARNAME=ANY に設定します。
DBMS テーブル名	VALIDVARNAME=ANY に設定します。
アイテムストア	VALIDMEMNAME=EXTEND に設定します。
SAS データセット名	VALIDMEMNAME=EXTEND に設定します。
SAS ビュー	VALIDMEMNAME=EXTEND に設定します。
ステートメントラベル	VALIDVARNAME=ANY に設定します。
変数	VALIDVARNAME=ANY に設定します。

名前リテラルが、特殊文字を格納する DBMS 列とテーブル名を表現し、SAS 名の各国文字を含めるために役立ちます。

次に、VAR ステートメントと名前リテラルの例を示します。

```
var 'a b'n;
```

次に、変数 A および B を含む VAR ステートメントの例を示します。

```
var a b;
```

SAS 名前リテラルの例

SAS 名前リテラルの例を次に示します。

- ```
  libname foo SAS/ACCESS-engine-name
              SAS/ACCESS-engine-connection-options;
  data foo.'My Table'n;
  ```
- ```
  data ‘Años de empleo’n.;
  ```
- ```
  data “August Purchases”n;
  ```
- ```
  input 'Bob''s Asset Number'n;
  ```
- ```
  input “Bob's Asset Number”n;
  ```
- ```
  input 'Amount Budgeted'n 'Amount Spent'n
        'Amount Difference'n;
  ```
- ```
  ‘Statement Label 1’n:
  ```

重要な制限事項

- 名前リテラルは、変数、ステートメントラベル、DBMS 列とテーブル名、SAS データセット、SAS ビュー、アイテムストアでのみ使用できます。
- SAS データセット名、SAS ビュー名、アイテムストアの名前リテラルは、VALIDMEMNAME=EXTEND の場合に使用できない文字を格納している場合、システムオプション VALIDMEMNAME=EXTEND を設定する必要があります。“VALIDMEMNAME= System Option” (*SAS System Options: Reference*)を参照してください。
- VALIDVARNAME=V7 で使用できない文字が変数、DBMS テーブル、または DBMS 列の名前リテラルに含まれている場合は、VALIDVARNAME=ANY を設定する必要があります。“VALIDVARNAME= System Option” (*SAS System Options: Reference*)を参照してください。
- パーセント記号(%)かアンパサンド(&)のどちらかを使用する場合は、SAS マクロ機能との相互作用を避けるために名前リテラルに一重引用符を使用する必要があります。
- SAS 命名規則に準拠しない文字が DBMS テーブルまたは DBMS 列の名前リテラルの中に含まれている場合は、SAS/ACCESS ソフトウェアの LIBNAME ステートメントのオプションを指定する必要があります。

  注: SAS/ACCESS ソフトウェアの LIBNAME ステートメントの詳細と例、および SAS 命名規則に従っていない DBMS テーブルの使い方については、*SAS/ACCESS for Relational Databases: Reference* を参照してください。
- 引用符で囲んだ文字列で、SAS は前置ブランクは保持して使用しますが、後置ブランクは無視して削除します。
- 閉じ引用符と n との間のブランクは、名前リテラルを指定する場合は無効です。
- VALIDVARNAME=ANY を設定する場合でも、V6 Engine は、ブランクが間に入っている名前をサポートしていません。

BY グループでの名前リテラルの使用

BY グループ処理の BY 変数として名前リテラルを指定し、対応する FIRST.または LAST.一時変数を参照する場合、2 レベルの変数名の FIRST.または LAST.部分を引用符で囲みます。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;
```

BY グループ処理と、一時変数 FIRST および LAST の作成方法の詳細については、“FIRST.variable と LAST.variable の識別” (443 ページ)および“How SAS Identifies the Beginning and End of a BY Group” (*SAS Statements: Reference*)を参照してください。

名前リテラルの使用時のエラーの回避

名前リテラルの作成でエラーを避ける方法については、“定数のよくあるエラーを避ける” (91 ページ)を参照してください。

SAS データセットと SAS 変数のデフォルトの命名規則の要約

VALIDMEMNAME システムオプションが COMPATIBLE に設定されており、かつ VALIDVARNAME システムオプションが V7 に設定されている(これが Base SAS のデフォルト設定である)場合の、SAS データセットと SAS 変数のデフォルトの命名規則の要約を次の表に示します。SAS Visual Analytics のような一部の SAS アプリケーションでは、SAS 変数および SAS データセットの命名規則を最も柔軟にするために、デフォルトで VALIDMEMNAME および VALIDVARNAME システムオプションが設定されます。これらの規則の要約については、表 3.4 (34 ページ)を参照してください。

表 3.3 *SAS データセットと SAS 変数のデフォルトの命名規則の要約*

データセット名 (VALIDMEMNAME=V7 の場合)	変数名 (VALIDVARNAME=COMPAT の場合)
• 長さが 32 文字を超えないこと • 特殊文字、ブランク、各国固有の文字は使用不可(アンダースコアは除く) • 最初の文字は、アルファベット(A–Z、a–z)またはアンダースコアで始まること • 大文字小文字の混在は許可されるが、大文字と小文字の組み合わせだけが異なる同じメンバ名を使用して、異なる変数を表現することはできない	• 長さが 32 文字を超えないこと • 特殊文字、ブランク、各国固有の文字は使用不可(アンダースコアは除く) • 最初の文字は、アルファベット(A–Z、a–z)またはアンダースコアで始まること • 大文字小文字の混在は許可されるが、大文字と小文字の組み合わせだけが異なる同じ変数名を使用して、異なる変数を表現することはできない

SAS データセットと SAS 変数の拡張命名規則の要約

高度なレベルの柔軟性が許可されている場合(すなわち、VALIDMEMNAME システムオプションが EXTEND に設定されており、かつ VALIDVARNAME システムオプションが ANY に設定されている場合)の、SAS データセット(テーブル)および SAS 変数(列)の命名規則の要約を次の表に示します。

表 3.4 SAS データセットと SAS 変数の拡張命名規則の要約

データセット名 (VALIDMEMNAME=EXTEND の場合)	変数名 (VALIDVARNAME=ANY の場合)
• 長さが 32 文字を超えないこと • 名前を名前リテラル(引用符で囲んだ文字列の後に文字 *n* が続くトークン)として指定すること • / \ * ?" < > | :-は指定できない • ブランクまたはピリオドでは開始できない • 大文字小文字の混在は許可されるが、大文字と小文字の組み合わせだけが異なる同じ名前を使用して、異なるデータセットを表現することはできない [1]	• 長さが 32 文字を超えないこと • 名前を名前リテラル(引用符で囲んだ文字列の後に文字 *n* が続くトークン)として指定すること • / \ * ?" < > | :-は指定できない • 大文字小文字の混在は許可されるが、大文字と小文字の組み合わせだけが異なる同じメンバ名を使用して、異なる変数を表現することはできない

1 UNIX 環境では、SAS は、すべて小文字で記述されたデータセット名のみを読み取る

4 章

SAS 変数

SAS 変数の定義

変数
: 変数には、プログラム内で使用するための文字値または数値を格納します。変数は、名前や種類(文字または数値)などの属性を持ちます。属性は、変数をどのように使用するかを識別して定義するものです。

文字変数
: アルファベット、数字(0 - 9)、その他の特殊文字を格納します。

数字変数
: 数値を浮動小数点数として格納します。日付や時刻なども含まれます。

数値精度
: 数値変数が動作環境に格納される精度です。

SAS 変数の属性

SAS 変数が持つ属性を、次の表に示します。

表 4.1 *変数の属性*

変数の属性	有効な値	デフォルト値
名前	有効な SAS 名。3 章, “SAS 言語のワードと命名規則について” (21 ページ)を参照してください。	なし
種類 *	数値および文字。	数値
長さ *	2 - 8 バイト。 ** 1 - 32,767 バイト(文字の場合)。	8 バイト(数値、文字の場合)
出力形式	次の文書を参照してください。 “Dictionary of Formats” (*SAS Formats and Informats: Reference*)	BEST12.BEST12.出力形式(数値の場合)、$w.出力形式(文字の場合)
入力形式	次の文書を参照してください。 “About Informats” (*SAS Formats and Informats: Reference*)	w.d 入力形式(数値の場合)、$w.入力形式(文字の場合)
ラベル	最大 256 文字。	なし

変数の属性	有効な値	デフォルト値
オブザベーション内の位置	1- n	なし
インデックスの種類	NONE、SIMPLE、COMPOSITE、BOTH	なし
拡張属性(ユーザー定義)	数値または文字	なし

* 変数の種類と長さは、明示的に定義されていない場合、その変数が DATA ステップ内で最初に出現するときに暗黙的に定義されます。

** 動作環境によっては、最小長が 2 バイトの場合も、3 バイトの場合もあります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

注: SAS 9.1 から、変数の最大数は 32,767 を超えることが可能になっています。実際の最大数は、環境やファイルの属性によって決まります。たとえば、すべての変数の合計の長さなどで決まり、最大ページサイズを超えることはできません。

CONTENTS プロシジャ、または次に挙げる SAS 関数を使用して、変数の属性に関する情報を取得できます。

名前
: 変数を識別します。変数名は SAS 命名規則に準拠している必要があります。規則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。表 3.3 (33 ページ)

  _N_、_ERROR_、_FILE_、_INFILE_、_MSG_、_IORC_、_CMD_という名前は、DATA ステップで自動的に生成される自動変数のために予約されています。SAS System では、先頭と末尾がアンダースコア(_)の自動変数をシステム側で変数名として使用するので、先頭と末尾がアンダースコア(_)の変数名を使用しないことをお勧めします。詳細については、“自動変数” (46 ページ) を参照してください。

  属性値を調べるには、VNAME 関数または VARNAME 関数を使用します。

種類
: 変数の種類は、数値か文字のどちらかになります。DATA ステップでは、変数は、文字であると指定しない限り、数値であると見なされます。数値は数字をさまざまな方法で読み込むことができ、浮動小数点形式で保存します。文字値には、アルファベット、数字、特殊文字を使用できます。長さは 1 - 32,767 文字です。

  変数の種類を調べるには、VTYPE 関数または VARTYPE 関数を使用します。

長さ
: 変数の値を SAS データセット内に格納するときに使用するバイト数を示します。LENGTH ステートメントを使用して、数値変数および文字変数の長さを明示的に設定できます。LENGTH ステートメントで指定した数値変数の長さは、出力データセット内の数値変数の長さにだけ影響します。プログラムデータベクトル内での処理中は、すべての数値変数の長さが 8 になります。LENGTH ステートメントで指定した文字変数の長さは、処理中の長さと、出力データセット内の長さの両方に影響します。

  INPUT ステートメントで、文字変数にデフォルト以外の長さを割り当てることができます。ATTRIB ステートメントでも、変数に長さを割り当てることができます。ある変数を割り当てステートメントの左辺に初めて記述した場合、その変数はその割り当てステートメントの左辺にある式の結果と同じ長さになります。

  変数の長さを調べるには、VLENGTH 関数または VARLEN 関数を使用します。

出力形式
: SAS System がデータ値を書き出す形式を示します。出力形式を指定しない場合、数値変数のデフォルトは BEST12.出力形式、文字変数のデフォルトは$w.出力形式です。FORMAT ステートメントまたは ATTRIB ステートメントを使用して、変数に

SAS 出力形式を割り当てることができます。FORMAT プロシジャを使用して、変数に対する独自の出力形式を作成することもできます。

変数の出力形式を調べるには、VFORMAT 関数または VARFMT 関数を使用します。

入力形式
: データ値の読み取り時に使用される命令を示します。入力形式を指定しない場合、数値変数のデフォルトは w.d 入力形式、文字変数のデフォルトは$w.入力形式です。INFORMAT ステートメントまたは ATTRIB ステートメントにより、変数に SAS 入力形式を割り当てることができます。FORMAT プロシジャを使用して、変数に対する独自の入力形式を作成することもできます。

  変数の入力形式を調べるには、VINFORMAT 関数または VARINFMT 関数を使用します。

ラベル
: 256 文字までのラベルを示します。変数ラベルは、データの管理やレポート作成に役立ちます。LABEL ステートメントまたは ATTRIB ステートメントにより、変数にラベルを割り当てることができます。

  変数のラベルを調べるには、VLABEL 関数または VARLABEL 関数を使用します。

オブザベーション内の位置
: DATA ステップ内で変数が何番目に定義されるかを示します。CONTENTS プロシジャを使用すると、SAS データセットのオブザベーション内での変数の位置がわかります。この情報は、次のような変数リストを使う場合に重要です。

```
var rent-phone;
```

  詳細については、“SAS 変数リスト” (47 ページ)を参照してください。

  SAS データセット内の変数の位置は、変数が SAS プロシジャの出力に表示される順序に影響します。(VAR ステートメントなどを使用して)プログラム内で順序を制御する場合は、この限りではありません。

  オブザベーションの位置を調べるには、VARNUM 関数を使用します。

インデックスの種類
: 変数が、データセットのインデックスの一部であるかどうかを示します。詳細については、“SAS インデックスについて” (632 ページ) を参照してください。

  インデックスの属性値を調べるには、CONTENTS プロシジャの OUT=オプションを使用して、出力データセットを作成します。出力データセットの IdxUsage 変数には、各変数に関して次の値のいずれかが含まれています。

表 4.2 *インデックスの種類属性値*

値	定義
NONE	変数にインデックスが付いていません。
SIMPLE	変数が単一インデックスの一部となります。
COMPOSITE	変数が、1 つ以上の複合インデックスの一部となります。
BOTH	変数が、単一インデックスと複合インデックスの両方の一部となります。

拡張属性
DATASETS プロシジャで XATTR ADD VAR ステートメントを使用して作成されるユーザー定義属性です。詳細については、“Extended Attributes” (*Base SAS Procedures Guide*)を参照してください。

変数の作成方法

概要

次に、DATA ステップで変数を作成する最も一般的な方法をいくつか挙げます。

- 割り当てステートメントを使用する。
- DATA ステップの INPUT ステートメントを用いてデータを読み込む。
- FORMAT ステートメントまたは INFORMAT ステートメントで新しい変数を指定する。
- LENGTH ステートメントで新しい変数を指定する。
- ATTRIB ステートメントで新しい変数を指定する。

注: このリストはすべてを網羅したものではありません。たとえば、SET、MERGE、MODIFY、UPDATE ステートメントでも変数を作成できます。

割り当てステートメントの使用

DATA ステップ内で、新しい変数を作成し、その変数に値を割り当てるには、初めて使用する変数を割り当てステートメントの左辺に記述します。変数の長さは、その変数が DATA ステップ内で最初に出現するときに決まります。新しい変数は、割り当てステートメントの右辺の式の値と同じ種類および長さになります。

変数の種類と長さが明示的に設定されていない場合は、次表の例のようなデフォルトの種類と長さが使用されます。

表 4.3 *明示的に設定されていない場合に使用される変数の種類と長さ*

式	例	X の種類	X の長さ	説明
数値変数	`length a 4;` `x=a;`	数値変数	8	数値のデフォルトの長さ(指定されない限り 8 バイト)
文字変数	`length a $ 4;` `x=a;`	文字変数	4	元の変数の長さ
文字リテラル	`x='ABC';` `x='ABCDE';`	文字変数	3	検出された最初の定数の長さ

式	例	X の種類	X の長さ	説明

変数の連結	`length a $ 4`
	`b $ 6`
	`c $ 2;`
	`x=a||b||c;`
		文字変数	12	すべての変数の長さの合計

変数と定数の連結	`length a $ 4;`
	`x=a||'CAT';`
	`x=a||'CATNIP';`
		文字変数	7	最初の割り当てステートメントで検出された変数と定数の長さの合計

初めて使用する変数が割り当てステートメントの右辺に記述されると、その変数は数値の欠損値になります。後続のステートメントで変数に値が入れられない場合、その変数が初期化されていないという NOTE メッセージが SAS ログに出力されます。SAS ログに NOTE メッセージを出力するかわりに、VARINITCHK=システムオプションを使用して、SAS ログに NOTE メッセージを書き込まない、すなわち、SAS ログに警告またはエラーメッセージを書き込む指定ができます。エラーが設定されると、DATA ステップは処理を停止します。詳細については、“VARINITCHK= System Option” (*SAS System Options: Reference*)を参照してください。

注: RETAIN ステートメントは、割り当てステートメントの後に指定された場合でも、変数を初期化し、変数に初期値を割り当てることができます。

DATA ステップの INPUT ステートメントを用いたデータの読み込み

INPUT ステートメントを使用して SAS System に生データを読み込む場合、その生データの編成方法に基づいて変数を定義します。INPUT ステートメントとともに次のいずれかの方法を使用して、生データに関する編成情報を SAS System に読み込みます。

- カラム入力
- リスト入力(通常のリスト入力またはフォーマット修飾子を使用したリスト入力)
- フォーマット入力
- 名前付き入力

それぞれの方法の使い方については、*SAS Formats and Informats: Reference* を参照してください。

次の例では、単純なリスト入力を使用して Gems という名前の SAS データセットを作成し、提供されたデータに基づいて 4 つの変数を定義しています。

```
data gems;
   input Name $ Color $ Carats Owner $;
   datalines;
emerald green 1 smith
sapphire blue 2 johnson
ruby red 1 clark
;
```

FORMAT ステートメントまたは INFORMAT ステートメントにおける新しい変数の指定

FORMAT ステートメントまたは INFORMAT ステートメントを使用すると、変数を作成した上で、その変数の出力形式または入力形式を指定できます。たとえば、次の FORMAT ステートメントは、Sales という名前の新しいデータセットに、6.2 出力形式で Sale_Price という名前の変数を作成します。

```
data sales;
   Sale_Price=49.99;
   format Sale_Price 6.2;
run;
```

SAS は、Sale_Price という名前で、長さが 8 の数値変数を作成します。

FORMAT ステートメントと INFORMAT ステートメントの使用方法については、*SAS Formats and Informats: Reference* を参照してください。

LENGTH ステートメントにおける新しい変数の指定

LENGTH ステートメントを使用すると、変数を作成した上で、その変数の長さを設定できます。次に例を示します。

```
data sales;
   length Salesperson $20;
run;
```

文字変数の場合、その変数を使用する最初のステートメントで、できるだけ長い値を使用する必要があります。これは、同じ DATA ステップ内の後続の LENGTH ステートメントでは、その長さを変更できないことによります。SAS 内の文字変数の最大長は、32,767 バイトです。数値変数の場合は、後続の LENGTH ステートメントで変数の長さを変更できます。

文字変数に値が割り当てられる場合、ターゲット変数の長さに合うように、必要に応じて値にブランクが埋め込まれたり、右辺の値が切り捨てられたりします。次のステートメントについて考えてみます。

```
length address1 address2 address3 $ 200;
address3=address1||address2;
```

Address3 の長さが 200 バイトなので、連結の最初の 200 バイト(Address1 の値)だけが Address3 に割り当てられます。このような場合、連結を実行する前に TRIM 関数を使用して Address1 から後置ブランクを削除することによって、問題を回避できることがあります。次に例を示します。

```
address3=trim(address1)||address2;
```

詳細については、“LENGTH Statement” (*SAS Statements: Reference*)を参照してください。

ATTRIB ステートメントにおける新しい変数の指定

ATTRIB ステートメントでは、次のオプションを使用することにより、既存の変数の変数属性を一度に 1 つ以上指定できます。

- FORMAT=
- INFORMAT=
- LABEL=

- LENGTH=

変数がまだ存在しない場合、FORMAT=、INFORMAT=、LENGTH=オプションの属性を 1 つまたは複数使用して、新しい変数を作成することができます。たとえば、次の DATA ステップでは、Lollipops というデータセットに FLAVOR という名前の変数を作成します。

```
data lollipops;
   Flavor="Cherry";
   attrib Flavor format=$10.;
run;
```

注: LABEL ステートメント、または ATTRIB ステートメントの LABEL=オプションの属性を単独で使用して新しい変数を作成することはできません。ラベルは、既存の変数だけに割り当てることができます。

詳細については、“ATTRIB Statement” (*SAS Statements: Reference*)を参照してください。

IN=データセットオプションの使用

IN=データセットオプションは、データセットからオブザベーションを読み込んだかどうかを示す特殊なブール変数を作成します。TRUE の場合、変数の値は 1 になり、FALSE の場合、変数の値は 0 になります。IN=データセットオプションは、DATA ステップ内の SET、MERGE、UPDATE ステートメントで使用できます。

次の例は、Old データセットと New データセットのマージを示します。ここでは、IN=オプションを使用して X という名前の変数が作成され、New データセットからオブザベーションが読み込まれたかどうかを示します。

```
data master missing;
   merge old new(in=x);
   by id;
   if x=0 then output missing;
   else output master;
run;
```

変数の種類の変換

数値変数として定義した変数に文字式の結果を割り当てると、SAS System は、式の文字結果を数値に変換してステートメントを実行しようとします。変換できない場合、SAS ログにメッセージが表示され、数値変数に欠損値が割り当てられ、自動変数 _ERROR_の値が 1 に設定されます。文字変数から数値変数へまたは数値変数から文字変数への自動変換の規則については、“数値と文字の自動変換” (92 ページ)を参照してください。

文字変数として定義した変数に数式の結果を割り当てると、SAS System は、式の数値結果を文字値に変換しようとします。SAS System は、BESTw.出力形式(w は文字変数の幅で、最大値は 32)を使用して、ステートメントを実行しようとします。使用する文字変数がその数字の文字表現を格納するのに十分な長さでない場合は、SAS ログにメッセージが表示され、文字変数にアスタリスク(*)が割り当てられます。値が小さすぎる場合は、エラーメッセージは表示されず、文字変数に文字 0 が割り当てられます。

ログ 4.1 *変数の種類の自動変換(SAS ログの一部)*

```
44 data _null_; 45 x= 3626885; 46 length y $ 4; 47 y=x; 48 put y; 49 run;
NOTE:Numeric values have been converted to character values at the places given
by:(Line):(Column).47:6 36E5 50   data _null_; 51      x1= 3626885; 52
length y1 $ 1; 53      y1=x1; 54      xs=0.000005; 55      length ys $ 1;
56      ys=xs; 57      put y1= ys=; 58   run; NOTE:Numeric values have been
converted to character values at the places given by:(Line):(Column).53:7   56:7
NOTE:Invalid character data, x1=3626885.00 , at line 53 column 7. y1=* ys=0
x1=3626885 y1=* xs=5E-6 ys=0 _ERROR_=1 _N_=1 NOTE:At least one W.D format was
too small for the number to be printed.The decimal may be shifted by the "BEST"
format.59   proc printto; run;
```

この例の最初の DATA ステップでは、変数 Y の値を指数表記で表現することによって、4 バイトのフィールドに収めることができます。2 番目の DATA ステップでは、変数 Y1 の値を 1 バイトのフィールドに収めることができないので、アスタリスク(*)が表示されます。

SAS 出力における変数値の位置調整

SAS 出力では、数値変数は右揃え、文字値は左揃えになります。出力形式を使用すると、位置調整をさらに制御できます。

ただし、SAS LISTING 出力では、割り当てステートメントで文字値を割り当てると、その値はステートメントで指定されたとおりに保存され、位置調整は行われません。アウトプット 4.1 (44 ページ)は、次のプログラムによって生成された文字値の位置調整を示しています。

```
ods listing;
options nodate;
data aircode;
   input city $ 1-13 WAC 15-17;
   length airport $ 30;

   if city='San Francisco' then airport='SFO';
      else if city='Paris' then airport='CDG';
      else if city='New York' then airport='JFK';
      else if city='Moscow' then airport='MOW';
      else if city='Melbourne' then airport='    MEB';

datalines;
San Francisco 67
Paris         427
New York      67
Moscow        770
Melbourne     802
;
proc print data=aircode;
run;
ods listing close;
```

この例では、次の LISTING 出力が生成されます。

アウトプット 4.1 SAS リスト出力における文字変数の位置調整

Obs	city	WAC	airport
1	San Francisco	67	SFO
2	Paris	427	CDG
3	New York	67	JFK
4	Moscow	770	MOW
5	Melbourne	802	MEB

HTML 出力では、割り当てステートメントで文字値を割り当てると、SAS は空白を無視して、通常通りに文字を左揃えにします。前述した同じ例で、HTML では次の出力になります。

アウトプット 4.2 SAS HTML 出力における文字変数の位置調整

The SAS System

Obs	city	WAC	airport
1	San Francisco	67	SFO
2	Paris	427	CDG
3	New York	67	JFK
4	Moscow	770	MOW
5	Melbourne	802	MEB

SAS 出力における変数の順番の変更

次の宣言ステートメントを使うと、SAS 出力に表示される変数の順番を制御できます。

- ARRAY
- ATTRIB
- FORMAT
- INFORMAT
- LENGTH
- RETAIN

これらのステートメントを機能させるためには、それらを次の宣言ステートメントのいずれかの前に記述する必要があります。

- SET
- MERGE
- UPDATE

位置に関連する変数のみをリストする必要があります。これらのステートメント内にリストされない変数は、元の位置を保持します。たとえば、データセット Sashelp.Class に変数 Name、Sex、Age、Height、Weight がこの順番で含まれている場合、変数 Height を先頭位置に移動するには、次の例に示すような LENGTH ステートメントを使用します。

例のコード 4.1 *Using the LENGTH Statement to Reorder Variables*

```
data Class1;
  length Height 3;     /* The LENGTH statement precedes the SET statement   */
  set Sashelp.Class;  /* and causes the variable Height to be placed first */
run;                                                    /* in the output  */
proc print data=Class1;
run;
```

アウトプット 4.3 *LENGTH ステートメントを使用した変数の順番の変更*

The SAS System

Obs	Height	Name	Sex	Age	Weight
1	69.0000	Alfred	M	14	112.5
2	56.5000	Alice	F	13	84.0
3	65.2969	Barbara	F	13	98.0
4	62.7969	Carol	F	14	102.5

変数の順番を変更する場合、RETAIN ステートメントが最もよく使用されます。これは、RETAIN ステートメントでは、その他の変数属性の指定が必要ないためです。RETAIN ステートメントは、データセットから読み込まれる既存の変数値の保持には影響しません。次の例では、RETAIN ステートメントを使用して、変数 Weight を、出力データセットの先頭にリストしています。

例のコード 4.2 *Using the RETAIN Statement to Reorder Variables*

```
data Class2;
  retain Weight;      /* The RETAIN statement precedes the SET statement   */
  set Sashelp.Class;  /* and causes the variable Weight to be placed first */
run;                                                    /* in the output  */
proc print data=Class2;
run;
```

アウトプット 4.4 RETAIN ステートメントを使用した変数の順番の変更

The SAS System

Obs	Weight	Name	Sex	Age	Height
1	112.5	Alfred	M	14	69.0
2	84.0	Alice	F	13	56.5
3	98.0	Barbara	F	13	65.3
4	102.5	Carol	F	14	62.8

自動変数

自動変数とは、DATA ステップまたは DATA ステップステートメントによって自動的に作成される変数です。これらの変数は、プログラムデータベクトルに追加されますが、作成されるデータセットには出力されません。自動変数の値は、DATA ステップの次の反復でも維持され、欠損値には設定されません。

特定のステートメントによって作成される自動変数については、各ステートメントの説明を参照してください。例については、“BY Statement” (*SAS Statements: Reference*)、“MODIFY Statement” (*SAS Statements: Reference*)、および“WINDOW Statement” (*SAS Statements: Reference*)を参照してください。

DATA ステップを開始するたびに、_N_と_ERROR_という 2 つの自動変数が作成されます。

_N_
: 最初は 1 に設定されます。DATA ステップが DATA ステートメントをループするたびに、自動変数_N_の値は 1 ずつ増加します。_N_の値は、DATA ステップの反復が開始された回数を表しています。

ERROR
: デフォルトでは 0 です。入力データエラー、変換エラー、算術エラー(0 による除算や浮動小数点のオーバーフロー)など、エラーが検出されると 1 に設定されます。この変数の値を使用して、データレコード内のエラーを見つけたり、SAS ログにエラーメッセージを表示したりできます。

  たとえば、次の 2 つのステートメントはいずれも、DATA ステップの各反復中に、入力エラーが検出された入力レコードの内容を SAS ログに表示します。

```
if _error_=1 then put _infile_;
if _error_ then put _infile_;
```

SAS 変数リスト

定義

SAS 変数リストは、ステートメントや関数の引数とする複数の変数名を列挙したもので、略記法を使用することができます。SAS System では、次の変数リストを使用できます。

- 番号付き範囲リスト
- 名前の範囲リスト
- 名前の接頭辞リスト
- 特殊な SAS 変数名リスト

番号付き範囲リスト以外は、SAS System が変数を追跡する順序で、変数リスト内の変数を参照します。SAS System は、アクティブな変数を、コンパイラが DATA ステップ内で検出する順序で追跡します。これは、変数が既存のデータセットから読み込まれるか、外部ファイルから読み込まれるか、DATA ステップ内で作成されるかには関係しません。番号付き範囲リストでは、任意の順序で作成された変数を参照できます。ただし、変数名の接頭辞が同じである必要があります。

変数を定義するステートメントやデータセットオプションで、変数リストを使用できます。変数リストは、既存のデータグループを参照できる簡単な方法を提供するので、SAS プログラムですべての変数を定義した後で、特に役立ちます。

注: RENAME=データセットオプションでは、番号付き範囲リストだけ使用できます。

番号付き範囲リスト

番号付き範囲リストでは、最後の文字が連続した数字で、その数字以外は同じ名前である一連の変数である必要があります。たとえば、次の 2 つのリストは同じ変数を参照します。

```
x1,x2,x3,...,xn
```

```
x1-xn
```

番号付き範囲リストでは、ユーザーが任意に決定した変数名に則って数字が連続している限り、どの数字から始めてどの数字で終わってもかまいません。

たとえば、数値変数に VAR1、VAR2 などの連続した名前を付けるとします。この場合、次のような INPUT ステートメントを記述できます。

```
input idnum name $ var1-var3;
```

文字変数 NAME は、略記法で指定されていないことに注意してください。

名前の範囲リスト

名前の範囲リストは、次の表に示すように変数定義の順序に依存します。

表 4.4 名前の範囲リスト

変数リスト	含まれている変数
x--a	変数定義の順番で、*x* から *a* までのすべての変数
x-numeric-a	*x* から *a* までのすべての数値変数
x-character-a	*x* から *a* までのすべての文字変数

CONTENTS プロシジャで VARNUM オプションを使用し、定義の順番に変数を出力できます。

たとえば、次のような INPUT ステートメントを実行するとします。

```
input idnum name $ weight pulse chins;
```

後続のステートメントで、次の変数リストを使用できます。

```
/* keeps only the numeric variables idnum, weight, and pulse */
keep idnum-numeric-pulse;

/* keeps the consecutive variables name, weight, and pulse */

keep name--pulse;
```

名前の接頭辞リスト

一部の SAS 関数および SAS ステートメントでは、名前の接頭辞リストを使用して、指定した文字列で始まるすべての変数を参照できます。次に例を示します。

```
sum(of Sales:)
```

この文字列は、SAS System に対して、Sales_Jan、Sales_Feb、Sales_Mar など、“Sales” で始まるすべての変数の合計を計算するよう指示しています。

特殊な SAS 変数名リスト

特殊な SAS 変数名リストには、次のものがあります。

NUMERIC
: 現在の DATA ステップで定義されているすべての数値変数を指定します。

CHARACTER
: 現在の DATA ステップで定義されているすべての文字変数を指定します。

ALL
: 現在の DATA ステップで定義されているすべての変数を指定します。

OF 演算子による変数リスト

定義

OF 演算子は、変数リストや配列を引数とする一部の SAS 関数の引数の前に記述します。OF 演算子を使用する SAS 関数の一般的な構文を次に示します。

function-name(*OF variable-list*) | (*OF array-name*)

OF 演算子を、引数が番号付き範囲リスト形式で指定されている関数で使用する場合には注意が必要です。たとえば、番号付き範囲リスト形式(x1 – x*n*)で指定された引数は、OF 演算子がそのリストの前に記述されている場合にのみ、値の範囲として読み込まれます。同じリストの前に OF 演算子が記述されていない場合、そのリストは SUM 関数により使用され、ハイフン(-)は減算記号として扱われます。この関数は、値の範囲の合計ではなく、変数の間の差を返します。

次の例では、OF 演算子を使用するかどうかにかかわらず、引数が番号付き範囲リストとして渡されます。最初の SUM 関数は **T=20** を返し、2 番目の SUM 関数は **T2=60** を返します。

```
data _null_;
  x1=30; x2=20; x3=10;
  T=sum(x1-x3);
  T2=sum(OF x1-x3);
  put T=; /*returns the difference between x1 and x3.*/
  Put T2=; /*returns the sum of the variable values from x1 to x3 (inclusive)*/
run;
```

注: 欠損値を含んでいるデータに対する減算や加算などの計算操作を含む引数を渡した場合、SAS は欠損値を返し、計算は行われません。

OF 演算子を使用して、配列名を関数に渡すこともできます。次に例を示します。

```
varA=mean(of array-name[*]);
```

OF 演算子での配列の使用については、“Using the OF Operator with Temporary Arrays” (*SAS Functions and CALL Routines: Reference*)を参照してください。

複数の変数リスト

複数の範囲リストが関数の引数として使用される場合、OF 演算子を使用することにより、ある変数リストを別の変数リストから区別することができます。複数の変数範囲リストを使用する場合、リスト全体の前に OF 演算子を記述して、各リストをブランクで区切ります。または、各リストの前に OF 演算子を記述して、各リストをカンマで区切ります。次に例を示します。

```
T=sum(OF x1-x3 y1-y3 z1-z3)
```

または

```
T=sum(OF x1-x3, OF y1-y3, OF z1-z3)
```

SAS で使用する変数リストの種類の詳細については、“SAS 変数リスト” (47 ページ)を参照してください。

OF 演算子を使用する SAS 関数のリストについては、表 11.1 (169 ページ)を参照してください。

OF 演算子で使用できる SAS 変数リストの種類を次の表に示します。

表 4.5 *OF 演算子で使用できる SAS 変数リスト*

種類	例	説明
名前の範囲リスト	Function(OF *x-character-a*)	*x* から *a* までのすべての文字変数に対して関数を実行します。
名前の接頭辞リスト	Function(OF x)	“x1”や“x2”など、“x”で始まるすべての変数に対して関数を実行します。

番号付き範囲リスト	Function(OF x1 – x*n*)	x1 から x*n* までの範囲の変数値に対して関数を実行します。[1]
配列	Function((OF *array-name*(*))	指定された配列に対して関数を実行します。[2]
特殊な SAS 変数名リスト	Function(OF _numeric_)	_numeric_ 変数(現在の DATA ステップで定義されているすべての数値変数)に対して関数を実行します。

SAS 関数と OF 演算子の詳細については、“SAS Functions and CALL Routines” (*SAS Functions and CALL Routines: Reference*)を参照してください。

変数の削除、保持、名前変更

ステートメントまたはデータセットオプションの使用

DROP、KEEP、および RENAME ステートメント、または DROP=、KEEP=、および RENAME=データセットオプションは、DATA ステップ内でどの変数の処理や出力を行うかを制御します。これらのステートメントおよびデータセットオプションを単独または組み合わせて使用して、必要な結果を得ることができます。SAS によって実行されるアクションは、次のいずれかのアクションを実行するかによって決まります。

- ステートメントまたはデータセットオプションのみを使用する場合、あるいはその両方を使用する場合。
- 入力データセットまたは出力データセットでデータセットオプションを指定する場合。

次の表に、DROP、KEEP、RENAME ステートメントと、DROP=、KEEP=、RENAME=データセットオプションの一般的な違いを示します。

表 4.6 *変数の削除、保持、名前変更を行うためのステートメントとデータセットオプション*

ステートメント	データセットオプション
出力データセットにのみ適用されます。	出力データセットまたは入力データセットに適用されます。
すべての出力データセットに影響します。	個々のデータセットに影響します。
DATA ステップでのみ使用できます。	DATA ステップと PROC ステップで使用できます。
DATA ステップ内の任意の場所に指定できます。	適用するデータセット名の直後に指定します。

1 番号付き範囲リストには、最後の文字が連続した数字で、その数字以外は同じ名前である一連の変数が含まれます。

2 array-name が一時配列である場合には、制限事項が存在します。“Using the OF Operator with Temporary Arrays” (*SAS Functions and CALL Routines: Reference*)を参照してください。

入力データセットと出力データセットの使用

変数がプログラムデータベクトルに読み込まれる前、あるいは新しい SAS データセットに書き出すときのどちらに、変数の削除、保持、名前変更を行うかを検討することも必要です。DROP、KEEP、RENAME ステートメントを使用すると、これらの処理は、常に変数が出力データセットに書き出されるときに実行されます。SAS データセットオプションの場合は、どの位置で使用するかによって、いつ処理が実行されるかが決まります。入力データセットで使用すると、変数がプログラムデータベクトルに読み込まれる前に、変数の削除、保持、名前変更が行われます。出力データセットで使用すると、変数が新しい SAS データセットに書き出されるときにそのデータセットオプションが適用されます。(DATA ステップ内では、入力データセットは、SET、MERGE、UPDATE ステートメントで指定します。出力データセットは、DATA ステートメントで指定します。)どの方法を使用するか決定する場合、次の点を考慮します。

- 変数を出力データセットに書き出さず、変数の処理を必要としない場合は、入力データセットオプションを使用して、DATA ステップから変数を削除する方が効率的です。
- DATA ステップ内で変数を処理する前に変数の名前を変更する場合は、入力データセット内で RENAME=データセットオプションを使用する必要があります。
- 出力データセットに対して処理を行う場合は、出力データセット内でステートメントまたはデータセットオプションのいずれかを使用できます。

次の表に、データセットオプションおよびステートメントを、入力データセットおよび出力データセットに指定した場合の処理をまとめます。表の最後の列は、変数が DATA ステップ内の処理に使用できるかどうかを示しています。変数名を変更する場合は、RENAME の項目を参照してください。

表 4.7 *変数の削除、保持、名前変更を行う場合の変数と変数名の状態*

指定場所	データセットオプションまたはステートメント	目的	変数または変数名の状態
入力データセット	DROP= KEEP=	変数を処理から除外する、または処理する変数を指定します。	除外した変数は DATA ステップ内の処理で使用できません。
	RENAME=	処理する前に変数名を変更します。	プログラムステートメントおよび出力データセットオプションでは、変更後の新しい変数名を使用します。他の入力データセットオプションでは、変更前の古い変数名を使用します。

指定場所	データセットオプションまたはステートメント	目的	変数または変数名の状態
出力データセット	DROP、KEEP	どの変数を出力データセットに書き出すかを指定します。	入力したすべての変数を処理に使用できます。
	RENAME	すべての出力データセット内の変数名を変更します。	プログラムステートメントでは変更前の古い変数名を使用します。出力データセットオプションでは変更後の新しい変数名を使用します。
	DROP= KEEP=	どの変数を個々の出力データセットに書き出すかを指定します。	入力したすべての変数を処理に使用できます。
	RENAME=	個々の出力データセット内の変数名を変更します。	プログラムステートメントおよび他の出力データセットオプションでは変更前の古い変数名を使用します。

適用の順序

プログラムで複数のデータセットオプション、またはデータセットオプションやステートメントの組み合わせを使用する必要がある場合、次の順序で変数の削除、保持、名前変更が適用されることを知っておくと役立ちます。

- まず、入力データセットのオプションが、SET、MERGE、UPDATE ステートメントにより左から右に評価されます。DROP=データセットオプションと KEEP=データセットオプションは、RENAME=データセットオプションの前に適用されます。
- 次に、DROP ステートメントと KEEP ステートメントが適用され、その後 RENAME ステートメントが適用されます。
- 最後に、出力データセットのオプションが、DATA ステートメント内で左から右に評価されます。DROP=データセットオプションと KEEP=データセットオプションは、RENAME=データセットオプションの前に適用されます。

変数の削除、保持、名前変更の例

次の例では、変数の削除、保持、名前変更を行うためのさまざまな方法を示しています。

- 次の例では、DROP=データセットオプション、RENAME=データセットオプション、INPUT 関数を使用して、変数 PopRank の種類を文字から数値に変換しています。新しい変数 PopRank を出力データセットに書き出すため、変数名 PopRank は処理の前に TempVar に変更されます。変数 TempVar が出力データセットから除外されていることと、プログラムステートメントで新しい変数名 TempVar が使用されていることを確認します。

```
data newstate(drop=tempvar);
   length poprank 8;
   set state(rename=(poprank=tempvar));
   poprank=input(tempvar,8.);
run;
```

- 次の例では、DROP ステートメントと DROP=データセットオプションを使用して、2 つの新しい SAS データセットへの変数の出力を制御しています。DROP ステートメントは、両方のデータセット(Corn と Bean)に適用されます。RENAME=データセットオプションを使用して、各データセットの出力変数 BeanWt および CornWt の変数名を変更する必要があります。

```
data corn(rename=(cornwt=yield) drop=beanwt)
     bean(rename=(beanwt=yield) drop=cornwt);
   set harvest;
   if crop='corn' then output corn;
   else if crop='bean' then output bean;
   drop crop;
run;
```

- 次の例では、DATA ステートメントのデータセットオプションと RENAME ステートメントをともに使用する方法を示しています。DROP=データセットオプションで新しい変数名 QTRTOT が使用されていることに注意してください。

```
data qtr1 qtr2 ytd(drop=qtrtot);
   set ytdsales;
   if qtr=1 then output qtr1;
   else if qtr=2 then output qtr2;
   else output ytd;
   rename total=qtrtot;
run;
```

変数値の暗号化

SAS 変数に対するカスタマイズされた暗号化および復号化アルゴリズム

SAS では、ENCRYPT=データセットオプションによる SAS データセットの暗号化が提供されますが、このオプションは、通常はデータセットレベルでのデータの暗号化に使用されます。SAS 変数レベルでデータを暗号化するには、DATA ステップ関数とロジックの組み合わせを使用すると、独自の暗号化および復号化アルゴリズムを作成できます。ただし、独自のアルゴリズムを作成する場合は、セキュアかつ非公開なだけでなく、データの暗号化と復号化の両方の方法も含むプログラムを作成することが重要です。

このセクションでは、変数の暗号化と復号化のためのさまざまな関数とメソッドを伴う DATA ステップを使用したサンプルプログラムが提供されます。

例 1:TRANSLATE 関数を使用した単純な 1 バイトから 1 バイトへのスワップ

最初のサンプルプログラムでは、TRANSLATE 関数での単純な 1 バイトから 1 バイトへのスワップを使用する方法が示されます。この方法は複雑ではないので、セキュアではないとも考えられます。しかしながら、ユーザーが TRANSLATE 引数 `from` および `to` に対する文字、特殊文字、または値のパターンを設計するので、独自の一意な暗号化アルゴリズムを作成できます。

TRANSLATE **from** 引数と TRANSLATE **to** 引数のどちらに対して記述される値も、順次、アルファベット順、数値順にする必要はありません。

次のサンプルコードでは、DATA ステップで長さ 8 文字以下の名前を読み取り、DO ループを使用して、TRANSLATE 関数と SUBSTR 関数を 1 バイトずつ処理します。

1 番目の DO ループでは暗号化値を作成し、2 番目の DO ループでは、順序を逆にして元の値を返すことによって復号化値を作成します。

例のコード 4.3 *A Simple 1-Byte-to-1-Byte Swap Using the TRANSLATE Function*

```
data sample1;
input @1 name $;
length encrypt decrypt $ 8;

/*ENCRYPT*/
do i = 1 to 8;
   encrypt=trim(encrypt)||translate(substr(name,i,1),
   '0123456789!@#$%^&*()-=,./?<','ABCDEFGHIJKLMNOPQRSTUVWXYZ');
end;

/*DECRYPT*/
do j = 1 to 8;
   decrypt=trim(decrypt)||translate(substr(encrypt,j,1),
   'ABCDEFGHIJKLMNOPQRSTUVWXYZ','0123456789!@#$%^&*()-=,./?<');
end;

drop i j;
datalines;
ROBERT
JOHN
GREG
;
proc print;
run;
```

次の出力は、例 1 の PROC PRINT の結果を示しています。

The SAS System

Obs	name	encrypt	decrypt
1	ROBERT	*%14*)	ROBERT
2	JOHN	9%7$	JOHN
3	GREG	6*46	GREG

例 2:TRANWRD 関数を使用した 1 バイトから 2 バイトへのスワップの使用

このサンプルプログラムでは、TRANWRD 関数での 1 バイト対 2 バイトのスワップを使用した値の暗号化方法が示されます。各 1 バイト文字が 2 桁の数字に置換されます。1 バイトから多バイト、または多バイトから 1 バイトに値を変更するには、TRANWRD 関数を使用して、変換される変数と同じ名前を結果変数に割り当てる必要があります。**from** 値に対して記述される値は、特殊な順序にする必要はありません。

次のサンプルコードでは、DATA ステップで長さ 6 文字以下の ID を読み取ります。ただし、これは 1 バイトから 2 バイトへの交換なので、変数に長さ 12 が割り当てられ、文字長が 2 倍になります。DO ループでは、TRANWRD 関数が 1 バイトずつ処理されます。

変更される値は文字 A、B、C、D、E、F ですが、これらは **from_1** 値としてランダムな順序で記述されます。**to_1** 値には開始値 21 が割り当てられます。ただし、これは、最後に割り当てられる値が 99 を超えて 3 桁の数字にならない限り、11 や 38 などその他の 2 桁の数字にすることもできます。

1 番目の DO ループでは暗号化値を作成し、2 番目の DO ループでは、順序を逆にして元の値を返すことによって復号化値を作成します。元の ID 変数の無効化を回避するために、TRANWRD 関数の前に新しい変数が ID 変数に割り当てられます。

例のコード 4.4 *Using a 1-Byte-to-2-Byte Swap with the TRANWRD Function*

```
data sample2;
input @1 id $12.;

/*ENCRYPT*/
encrypt=id;
i=21;
do from_1 = "C","F","E","A","D","B";
  to_1=put(i,2.);
  encrypt=tranwrd(encrypt,from_1,to_1);
  i+1;
end;

/*DECRYPT*/
decrypt=encrypt;
j=21;
do to_2 = "C","F","E","A","D","B";
  from_2=put(j,2.);
  decrypt=tranwrd(decrypt,from_2,to_2);
  j+1;
end;
drop i j to_1 from_1 to_2 from_2;

datalines;
ABCDEF
FEDC
ACE
BDFA
CAFDEB
BADCF
ABC
;
proc print;
run;
```

次の出力は、例 2 の PROC PRINT の結果を示しています。

The SAS System

Obs	id	encrypt	decrypt
1	ABCDEF	242621252322	ABCDEF
2	FEDC	22232521	FEDC
3	ACE	242123	ACE
4	BDFA	26252224	BDFA
5	CAFDEB	212422252326	CAFDEB
6	BADCF	2624252122	BADCF
7	ABC	242621	ABC

例 3:異なる関数を使用して数値を文字列として暗号化する

このサンプルプログラムでは、数値を暗号化し、3 回ごとに異なる文字を使用して文字値を作成する方法を示します。この方法では、PUT、SUBSTR、INDEXC、TRANSLATE、CATS、INPUT 関数に加えて配列処理も使用します。

プログラムでは、2 つの DATA ステップを使用します。一方で値を暗号化し、もう一方で逆の処理を行って値を復号化します。必要に応じて、DATA ステップの暗号化と復号化を 1 つの DATA ステップにマージすることもできます。

1 番目の DATA ステップでは、5 桁以下の数値を読み取ります。数値変数が文字変数に変換され、5 つの別々の値に分割されます。

4 つの ARRAY ステートメントが使用されます。1 番目の配列で **from** 値を設定し、2 番目で **to** 値を設定し、3 番目で 5 つの別々の数値を保持し、4 番目で 5 つの新しい別々の暗号化値を保持します。

from および **to** 配列はそれぞれ 3 つの要素で作成されます。**from** ARRAY では、3 つの要素すべてに同じ数字列が割り当てられ、**to** ARRAY では、3 つの要素のそれぞれに異なる文字列が割り当てられて 3 回ごとの循環パターンが構築されます。

PUT 関数では、数値が文字値に変換されます。

1 番目の DO ループでは、SUBSTR 関数を使用して値を 5 つの別々の値に分割し、それぞれを古い ARRAY に割り当てます。2 番目の DO ループでは、各値を変換します。これには INDEXC 関数を使用して **from** ARRAY で元の数字を探し、見つかった場合は、**from** ARRAY を使用して値を変換し、3 回ごとに要素のリストを循環します。暗号化値は、CATS 関数を使用して 5 つの変換値を連結することによって作成されます。

後述する例の 2 つの DATA ステップを比較すると、**to** 配列と **from** 配列の値が逆になっていることがわかります。これは、2 番目の DATA ステップで、1 番目の DATA ステップで行われた暗号化を逆にし、値を変換して元の値に戻しているためです。

値の暗号化に使用されたのと同じプロセスが、値の復号化でも使用されます。唯一の違いは、暗号化変数は SUBSTR 関数に渡され、最終復号化変数は CATS 関数に続く INPUT 関数に渡されることです。これは最終値を数値にするために行われます。

例のコード 4.5 *Using Different Functions to Encrypt Numeric Values into Character Strings*

```
data sample3;
  input num;
  array from(3) $ 10 from1-from3 ('0123456789','0123456789','0123456789');
```

```
  array to(3) $ 10 to1-to3 ('ABCDEFGHIJ','KLMNOPQRST','UVWXYZABCD');
  array old(5) $ old1-old5;
  array new(5) $ new1-new5;
  char_num=put(num,5.);
  do i = 1 to 5;
    old(i)=substr(char_num,i,1);
  end;
  j=1;
  do k = 1 to 5;
    if indexc(old(k),from(j)) > 0 then do;
      new(k)=translate(old(k),to(j),from(j));
      j+1;
      if j=4 then j=1;
    end;
  end;
  encrypt_num=cats(of new1-new5);
  keep num encrypt_num;
  datalines;
  12345
  70707
  99
  1111
  ;
run;

data sample3;
  set sample3;
  array to(3) $ 10 to1-to3 ('0123456789','0123456789','0123456789');
  array from(3) $ 10 from1-from3 ('ABCDEFGHIJ','KLMNOPQRST','UVWXYZABCD');
  array old(5) $ old1-old5;
  array new(5) $ new1-new5;
  do i = 1 to 5;
    old(i)=substr(encrypt_num,i,1);
  end;
  j=1;
  do k = 1 to 5;
    if indexc(old(k),from(j)) > 0 then do;
      new(k)=translate(old(k),to(j),from(j));
      j+1;
      if j=4 then j=1;
    end;
  end;
  decrypt_num=input(cats(of new1-new5),5.);
  keep num encrypt_num decrypt_num;
run;

proc print;
run;
```

次の出力は、例 3 の PROC PRINT の結果を示しています。

The SAS System

Obs	num	encrypt_num	decrypt_num
1	12345	BMXEP	12345
2	70707	HKBAR	70707
3	99	JT	99
4	1111	BLVB	1111

SAS における数値の正確さ

概要

2 進法であれ 10 進法であれ、いかなる記数法においても、数値を表現する精度には限界があります。その結果、近似値が必要になります。たとえば、10 進法では、分数 `1/3` を有限小数値として完全に表現することはできません。それには無限に繰り返される数字 (`.333...`) が含まれるからです。コンピュータ上では、有限精度が原因で、この数字の近似値を求める必要があります。数値精度とは、数字の近似や表現の正確さのことです。

計算時には、ソフトウェアアプリケーションで、有限精度やマシンハードウェアの限界による数値精度誤差が特に発生しやすくなります。コンピュータは、有限の記憶容量を持つ有限のマシンなので、無限の数の集合を完全に正確に表現することはできません。

コンピュータが人間とは異なる記数法を使用するという事実が、この問題をさらに複雑にします。10 進無限精度演算が人間の計算の基準ですが、コンピュータは値の有限 2 進表現および有限精度演算を使用します。この表現は、多くの計算に適していることが実証されています。それにもかかわらず、問題によっては、ハードウェアの提供範囲よりも幅広い拡張精度が必要になる場合があります。その場合、表現と演算はほとんどソフトウェアで行われ、ハードウェア演算と比べると非常に遅くなります。

さらに、コンピュータでは人間中心のソフトウェアインターフェイスによって 10 進数と 10 進演算の使用が可能ですが、最終的にすべての数字とデータはバイナリ形式に変換されて、コンピュータで内部的に格納および処理されます。この 2 つの記数法間(10 進法から 2 進法)の変換において、精度に影響が及び、丸め誤差がもたらされます。

2 進数での切り捨て

無限繰り返し表現を含む 10 進値が存在するのと同様に、無限繰り返し表現を含む 2 進値も存在します。ただし、10 進法で不正確な数が、2 進法でも同じように不正確な数であるとは限りません。

たとえば、10 進値 `1/10` には有限 10 進表現(0.1)がありますが、2 進法ではこれが無限繰り返し表現になります。2 進法では、値は次のように変換されます。

```
0.0001100110011001100110011 ...
```

ここでは、パターン `0011` が無限に繰り返されます。その結果、値は、コンピュータでの格納時に丸められます。

SAS で不正確な数の計算と比較を実行すると、予期しない結果につながる可能性があります。最も単純な計算でさえも、間違った結論に至る可能性があるのです。ハードウェアが、10 進法で明白かつ予期どおりと思われた対象に常に適合可能とは限りません。

たとえば、10 進演算では、式 `(3 x 0.1)` は `0.3` に等しいと推測されるので、`(3 x 0.1)` と `(0.3)` の差は必ず 0 になると考えられます。10 進値 0.1 および 0.3 には正確な 2 進表現がないため、この等価性は 2 進演算にはあてはまりません。例のコード 4.6 (59 ページ) で説明されているように、SAS プログラムでこの 2 値の差を計算した場合、結果は 0 になりません。

例では、SAS で、変数 `point_three` と `three_times_point_three` をそれぞれ `0.3` と `(3 x 0.1)` に設定します。その後、2 つの値を比較するために、一方から他方を引いて、その結果を SAS ログに書き込みます。

例のコード 4.6 *Comparing Imprecise Values in SAS*

```
data a;
  point_three=0.3;
  three_times_point_one= 3 * 0.1;
  difference= point_three - three_times_point_one;
put 'The difference is ' difference;
run;
```

アウトプット 4.5 *SAS で不正確な値を比較するためのログ出力*

```
1    data a;
2      point_three=0.3;
3      three_times_point_one=3 * 0.1;
4      difference=point_three - three_times_point_one;
5      put 'The difference is ' difference;
6    run;

The difference is -5.55112E-17
NOTE: The data set WORK.A has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.99 seconds
      cpu time            0.12 seconds
```

ログ出力では、`(3 x 0.1) - 0.3` が、10 進演算と違って `0` に等しくならないことが示されます。これは、変数"difference"が、丸め値(すなわち無限に繰り返す 2 進値)に対して実行された計算の結果であるためです。

等値の 2 進数が無限に繰り返す 2 進数である小数は多数存在するので、10 進法の一般的な有理数からの結果を解釈する際には注意してください。どちらの記数法でも問題ない有理数も存在します。たとえば、1/2 は、10 進法と 2 進法のどちらでも有限表現が可能です。

このように単純な計算でなぜ間違いが起こり得るのか、また、どうして数が範囲外になり得るのかについて理解を深めるには、SAS での 2 進数の格納方法について詳細を理解することが重要です。

SAS での数値の格納法

最大整数サイズ

SAS は、特に指定しない限り、すべての数値を 8 バイトの記憶域内に格納します。これは、値が 8 桁に制限されているということではなく、値を格納するために 8 バイトが割り当てられるということを意味します。前のセクションでは、非整数値(小数)の格納法によっては精度の問題につながる可能性があることを学習しました。けれども、整数の操作時に、大きさ(すなわち範囲)の問題に遭遇する可能性もあります。

どんなコンピュータにも、整数の絶対値をどこまで大きくできるかには制限があります。SAS では、この最大整数値は 2 つの要因によって決まります。

- 変数の格納用に明示的に指定したバイト数(LENGTH ステートメントを使用)
- SAS が稼動している動作環境

記憶バイト数を明示的に指定しなかった場合、SAS ではデフォルト長の 8 バイトが使用され、最大整数は、どのオペレーティングシステムを使用しているかによってのみ決まります。

次の表に、メインフレーム、UNIX、および Windows 動作環境で SAS 変数によって確実に格納可能な最大整数を示します。

表 4.8 *指定長で安全に格納できる最大整数*

変数長が次に等しい場合	最大整数 z/OS	最大整数 Windows/UNIX
2	256	n/a
3	65,536	8,192
4	16,777,216	2,097,152
5	4,294,967,296	536,870,912
6	1,099,511,627,776	137,438,953,472
7	281,474,946,710,656	35,184,372,088,832
8 (デフォルト)	72,057,594,037,927,936	9,007,199,254,740,992

この表を見る場合の注意点を次に示します。

- Windows および UNIX オペレーティングシステムでの SAS 変数の最小長は 3 バイトで、最大長は 8 バイトです。IBM メインフレームでは、SAS 変数の最小長は 2 バイトで、最大長は 8 バイトです。
- 変数の長さが増えると、確実に表現できる整数サイズも増えます。
- 任意の与えられた変数長に関して、最大整数はホストごとに異なります。これは、メインフレームでは浮動小数点数の格納についての指定が UNIX および PC マシンとは異なるためです。
- 実数は常に完全な 8 バイトの記憶域に格納されます。LENGTH ステートメントを使用して変数の長さを減らすことによってディスク領域を節約したい場合、実行はできますが、対象は値が整数である変数のみです。変数の長さを調整する場合、値が、指定長に対して許可される最大整数以下であることを確認してください。

  たとえば、UNIX 動作環境では、数値変数の値が常に`-8192 and 8192` の間の整数であれば、数字を格納するために長さ 3 を安全に指定できます。

```
data myData;
  length num 3;
     num=8000;
run;
```

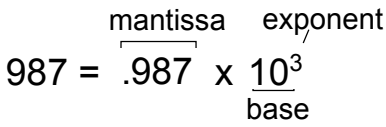
注意:

完全 8 バイトを使用して、実数を含む変数を格納します。

浮動小数点表現

SAS では、数値を 8 バイトのデータで格納します。数の格納方法に加えて、その格納に使用可能な空き領域も、数値の正確さに影響します。2 進数を内部格納するにはさまざまな方法がありますが、SAS では*浮動小数点表現*を使用して数値を格納します。浮動小数点表現では、広範囲にわたる値(非常に大きな数や非常に小さな数)が適度に正確な数値でサポートされます。

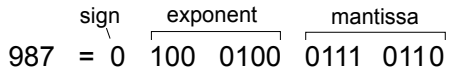
浮動小数点表現は*指数表記*に類似しているので、すでに精通しているユーザーもいることでしょう。指数表記と浮動小数点表現のどちらにおいても、各数は*仮数*、*基数*、および*指数*として表現されます。



- *仮数*は、基数を掛けられる数です。例では、仮数は `.987` です。
- *基数*は、累乗される数です。例では、基数は `10` です。
- *指数*は、基数を累乗する数です。例では、指数は `3` です。

指数表記と浮動小数点表現の大きな違いは、指数表記では、基数が 10 であることです。浮動小数点表現では、ほとんどのオペレーティングシステムで、基数は 2 か 16 です。どちらを使用するかはシステムによって異なります。

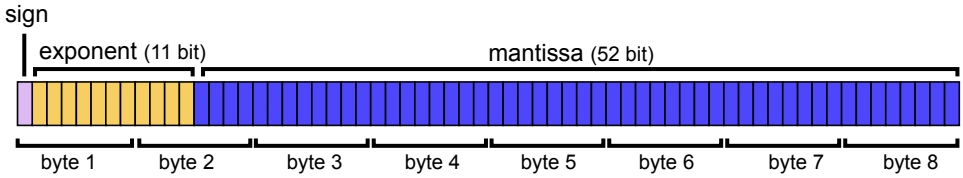
次の図は、IEEE 754 の 2 進浮動小数点形式で書き込まれた 10 進値 `987` を示します。これは小さな値なので、丸める必要はありません。



2 進浮動小数点数を格納するために、コンピュータでは、*交換形式*または*バイトレイアウト*と呼ばれる標準形式が使用されます。浮動小数点数の各部が標準化された方法で表現されるように、バイトレイアウトは、ビット列のグループ化およびオーダリングの標準的方法である左から右です。浮動小数点値の各部(符号、指数、仮数)には、列内の特定数のビットと、列内の特定位置が割り当てられます。これにより、効率的かつコンパクトな形式での浮動小数点データの交換が可能になります。

図 4.1 (61 ページ) では、倍精度 2 進浮動小数点数のバイトレイアウトが示されます。このレイアウトでは、最初のビットを使用して数の符号をエンコードし、次の 11 ビットで指数のエンコード、最後の 52 ビットで仮数のエンコードを行います。符号ビットが 1 の場合、その数は負であり、符号ビットが 0 の場合、その数は正となります。

図 4.1 *倍精度 2 進浮動小数点数のバイトレイアウト*



ホストコンピュータが異なれば、浮動小数点表現の形式や指定も異なる可能性があります。SAS System が稼動するすべてのプラットフォームでは、8 バイト浮動小数点表現が使用されます。

精度と大きさ

(丸めなしで)正確に表現できる最大整数値は、基数と、指数に割り当てられたビット数によって決まります。精度は、仮数に対して割り当てられたビット数によって決定されます。オペレーティングシステムが桁を切り捨てるのか丸めるのかは、表現のエラーに影響します。

SAS では、LENGTH ステートメントを使用して仮数ビット数を減らし、切り捨てられた浮動小数点数を格納します。次の表に、IBM メインフレームと IEEE 規格での浮動小数点形式の違いをいくつか示します。IEEE 規格は、Windows および UNIX オペレーティングシステムで使用されます。

表 4.9 浮動小数点形式の IBM および IEEE 規格

指定	IBM メインフレーム	IEEE 規格 (Windows/UNIX)	影響
基数	16	2	大きさ
指数ビット	7	11	大きさ
仮数ビット	56	52	精度
丸めまたは切り捨て	切り捨て	丸め	精度
指数のバイアス	64	1023	

次の箇条書き項目では、前述の表について詳細に説明します。

- **基数 16** – 数字 0-9 および文字 A-F (値 10-15 を表現するため)を使用します。

  たとえば、10 進値 `3000` を 16 進値に変換するには、基数 16 の記数法を使用します。

基数 16						
16^7	...	16^4	16^3	16^2	16^1	16^0
268,435,456	...	65,536	4096	256	16	1

$$3000 = (\mathbf{B} \times 16^2) + (\mathbf{B} \times 16^1) + (\mathbf{8} \times 16^0)$$
$$= (\mathbf{11} \times 256) + (\mathbf{11} \times 256) + (\mathbf{8} \times 1)$$

  そのため、値 3000 は、16 進法で `BB8` として表現されます。

- **基数 2** – 数字 0 および 1 を使用します。

  たとえば、10 進値 184 を 2 進値に変換するには、基数 2 の記数法を使用します。

基数 2						
2^7	...	2^4	2^3	2^2	2^1	2^0
128	...	16	8	4	2	1

$$184 = (\mathbf{1} \times 2^7) + (\mathbf{0} \times 2^6) + (\mathbf{1} \times 2^5) + (\mathbf{1} \times 2^4) + (\mathbf{1} \times 2^3) + (\mathbf{0} \times 2^2) + (\mathbf{0} \times 2^1) + (\mathbf{0} \times 2^0)$$
$$= 128 + 0 + 32 + 16 + 8 + 0 + 0 + 0$$

そのため、値 184 は、2 進法で `10111000` として表現されます。

- **指数ビット** – 指数を格納するために確保されるビット数であり、これにより格納できる数の大きさが決定されます。指数ビットの数は、オペレーティングシステム間で異なります。IEEE 準拠のシステムでは、指数により多くのビットを使用するため、より大きくなります。
- **仮数ビット** – 仮数を格納するために確保されるビット数であり、これにより数の精度が決定されます。メインフレームでは仮数により多くのビットが確保されているため、PC よりも高い精度が得られます。
- **丸めまたは切り捨て** – 2 桁以上の数の取り扱いに使用するために選択される変換方法。仮数には 2 つの 16 進数字のための余地しかないため、2 桁以上の数を取り扱う方法を採用する必要があります。1 つの方法は、格納できる長さで値を切り捨てることです。この方法は IBM メインフレームで採用されています。

  もう1つの方法は、格納できない数値に基づいて値を丸めることです。この方法は、VAX/VMS や IEEE 準拠システムで採用されています。どちらの方法も正確な値を表現できないため、このジレンマを解決する方法に優劣はありません。

  SAS では、LENGTH ステートメントは、仮数ビットの数を切り捨てることによって機能します。切り捨てられた長さの影響については、“値比較時の TRUNC 関数の使用” (74 ページ)を参照してください。
- **バイアス** – 0 を表すことによって正と負の両方の指数を表現できるようにするための方法です。バイアスを使用しないと、指数に追加の符号ビットを割り当てなければなりません。たとえば、システムがバイアス 64 を使用する場合、値 66 の指数部は+2 の指数を表し、61 の指数部は-3 の指数を表します。

IEEE 規格を使用した浮動小数点表現

浮動小数点演算の IEEE 規格は、IEEE (Institute of Electrical and Electronic Engineers)によって作成された浮動小数点計算の技術規格です。この規格によって、コンピュータによる浮動小数点表現での数の格納方法が定義されます。浮動小数点数の IEEE 規格は、Windows や UNIX を含む多くのオペレーティングシステムで使用されています。

IEEE プラットフォームでは同じ一連の仕様が使用されますが、コンパイラの違いや数学ライブラリの違いが原因で、場合によってはコンピュータ間でそれぞれ異なる結果が出ることもあります。また、IEEE 規格では、規格の実装方法の差異がある程度許容されているため、同じ規格に準拠していても、異なるプラットフォームでの計算実行には違いがある場合があります。各オペレーティングシステムが計算の実行に使用する基になる命令が多少異なるため、ホストで異なる結果が生まれることもあります。

計算の実行に標準的な方法はありません。すべてのオペレーティングシステムが、可能な限り正確に数を計算しようとします。浮動小数点表現の構成要素が異なるオペレーティングシステム間で多少異なる結果が出るのは珍しいことではありません。たとえば、z/OS と Windows のオペレーティングシステム間、および z/OS と UNIX のオペレーティングシステム間には違いがあります。

倍精度の浮動小数点数の IEEE 規格では、基数 2 でバイアス 1023 の 11 ビット指数が指定されます。つまり、IBM メインフレーム表現よりも大きくなりますが、かわりに仮数が 3 ビット少なくなることもあります。値 1 は、IEEE 規格では次のように表現されます。

```
3F F0 00 00 00 00 00 00
```

Windows プラットフォームでは、プロセッサが拡張実数精度で計算を実行します。すなわち、基本形式(仮数用に 52 ビットと指数用に 11 ビット)での数値の格納に 64 ビットが使用されるかわりに、仮数用に 12 追加ビットと指数用に 4 追加ビットの 16 追加ビットが存在します。SAS での数値変数の最大幅は 8 バイトなので、数値が 80 ビット(10 バイト)で格納されることはありません。これは単に、数値が 64 ビットメモリスロットに戻される前に、プロセッサが 80 ビットを使用して数値を表現することを意味します。中間計算を 80 ビットで行えることが、最終解答の一部に影響します。

Windows では、これにより、UNIX などのオペレーティングシステムで使用される基本 IEEE 浮動小数点形式よりも大きい数の格納が可能になります。同じ IEEE 規格を使用するオペレーティングシステムと多少異なる値が見つかることがある理由の 1 つはこれです。*拡張精度*形式では、基本浮動小数点形式よりも高い精度と広い指数範囲が提供されます。

Windows での浮動小数点表現

格納形式

Windows 環境における 64 ビット倍精度数のバイトレイアウトは次のとおりです。

S E E E E E E E 1 バイト目	E E E E M M M M 2 バイト目	M M M M M M M M 3 バイト目	M M M M M M M M 4 バイト目
M M M M M M M M 5 バイト目	M M M M M M M M 6 バイト目	M M M M M M M M 7 バイト目	M M M M M M M M 8 バイト目

このデータ表現は、次に解説するように、データのバイトに対応しており、各文字が 1 ビットを示しています。

- 1 バイト目の S は、数字の符号ビットです。符号ビットの値 0 は、正の数値を表します。
- 2 バイト目から 8 バイト目の残りの文字 M は、仮数のビットを表します。仮数の左端のビットの前に暗黙の基数点があります。したがって、仮数は必ず 1 より小さくなります。基数点という用語は、小数点の代わりに使用されます。これは、小数点は 10 進数(基数 10)を扱うことを意味しますが、10 進数を扱うとは限らないためです。基数点は、小数点の原型と見なすことができます。

指数には、基数が割り当てられています。これを、指数が表現される基数と混同しないでください。指数は、必ずバイナリ形式で表現されます。指数は、仮数に基数を何回掛ける必要があるかを特定するために使用されます。

変換の例

この例では、10 進値 255.75 の浮動小数点表現への変換プロセスを示します。

1. 基数 2 の記数法を使用して、2 進法で値 255.75 を書き出します。

   注: 仮数の各ビットは、分子が 1 で分母が 2 の累乗である分数を表します。つまり、仮数は、2 分の 1、4 分の 1、8 分の 1 などの一連の分数の合計となります。したがって、浮動小数点数を正確に表現するには、前述のような合計として表現する必要があります。

基数 2										
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	$.2^{-1}$	2^{-2}
	128	64	32	16	8	4	2	1	1/2	1/4
255.75 =	1 x 2^7	1 x 2^6	1 x 2^5	1 x 2^4	1 x 2^3	1 x 2^2	1 x 2^1	1 x 2^0	1 x 2^{-1}	1 x 2^{-2}

$$255.75 = (\mathbf{1} \times 2^7) + (\mathbf{1} \times 2^6) + (\mathbf{1} \times 2^5) + (\mathbf{1} \times 2^4) + (\mathbf{1} \times 2^3) + (\mathbf{1} \times 2^2) + (\mathbf{1} \times 2^1) + (\mathbf{1} \times 2^0) + (\mathbf{1} \times 2^{-1}) + (\mathbf{1} \times 2^{-2})$$

decimal point

そのため、値 255.75 は、バイナリ形式で 1111 1111.11 として表現されます。

2. この小数を、左側が 1 桁だけになるまで移動します。この処理は、値の正規化と呼ばれます。指数表記での値の*正規化*は、仮数の絶対値が 1 以上 10 未満になるように指数が選択されるようにする処理です。この数の場合、小数点を 7 桁移動します。

   ```
   1.111 1111 11
   ```

   小数点を 7 桁移動したので、指数は 7 になります。

3. バイアスは 1023 なので、1023 に 7 を足すと、次の値になります。

   ```
   1030
   ```

4. 基数 16 の記数法を使用して、10 進値 `1030` を 16 進値に変換します。

基数 16						
16^7	...	16^4	16^3	16^2	16^1	16^0
268,435,456	...	65,536	4096	256	16	1

$$\begin{aligned} 1030 &= (\mathbf{4} \times 16^2) + (\mathbf{0} \times 16^1) + (\mathbf{6} \times 16^0) \\ &= 1024 + 0 + 6 \end{aligned}$$

1030 に対する変換済み 16 進値は、最終結果の指数部に置かれます。

5. 406 のバイナリ形式への変換

   ```
   0100  0000  0110
     4     0     6
   ```

   変換する値が負になった場合は、最初のビットを 1 に変更します。

   ```
   1100  0000  0110
   ```

   これを 16 進法で変換すると次のようになります。

   ```
   C     0     6
   ```

6. 前述のステップ 2 で、最初の数字と小数(暗黙に定義された 1 ビット)を削除します。

   ```
   11111111
   ```

7. これをニブル(半バイト)に分割すると、次のようになります。

```
1111 1111 1
```

8. 最後に完全なニブルを含めるには、4 ビットを完全にするのに十分なゼロを追加します。

```
 1111 1111 1000
```

9.

```
1111 1111 1000
```

を等値の 16 進数に変換すると、仮数部が得られます。

```
1111  1111  1000
 F     F     8
```

255.75 の最終的な浮動小数点表現は、次のようになります。

```
406F F800 0000 0000
```

-255.75 の最終的な浮動小数点表現は、次のようになります。

```
C06F F800 0000 0000
```

この例では、最初の **255.75** は、都合よく、2 進法と 16 進法の両方で丸めなしで表現できる有限 2 進値に変換されます。次のセクションでは、浮動小数点表現では正確に表現できない 10 進数の変換プロセスを示します。

IBM メインフレームでの浮動小数点表現

格納形式

SAS for z/OS は、次のような従来の IBM メインフレームの浮動小数点表現を使用します。

S E E E E E E E	M M M M M M M M	M M M M M M M M	M M M M M M M M
1 バイト目	2 バイト目	3 バイト目	4 バイト目

M M M M M M M M	M M M M M M M M	M M M M M M M M	M M M M M M M M
5 バイト目	6 バイト目	7 バイト目	8 バイト目

このデータ表現は、次に解説するように、データのバイトに対応しており、各文字が 1 ビットを示しています。

- 1 バイト目の S は、数字の符号ビットです。符号ビットの値 0 は、正の数値を表します。
- 1 バイト目の 7 つの E は、指数部と呼ばれるバイナリ整数を表します。指数部は、符号付き指数を表し、実際の指数にバイアスを加えることによって取得されます。バイアスは、0 を表すことによって正と負の両方の指数を表現できるようにするための方法です。バイアスを使用しないと、指数に追加の符号ビットを割り当てなければなりません。たとえば、システムがバイアス 64 を使用する場合、値 66 の指数部は+2 の指数を表し、61 の指数部は-3 の指数を表します。
- 2 バイト目から 8 バイト目の残りの文字 M は、仮数のビットを表します。仮数の左端のビットの前に暗黙の基数点があります。したがって、仮数は必ず 1 より小さくなります。基数点という用語は、小数点の代わりに使用されます。これは、小数点

は 10 進数(基数 10)を扱うことを意味しますが、10 進数を扱うとは限らないためです。基数点は、小数点の原型と見なすことができます。

変換の例

次の例では、10 進値 **512.1** の 16 進浮動小数点表現への変換プロセスを示します。この例では、10 進法で正確に表現できない値を 16 進浮動小数点式で正確に表現することはできないということを示します。

1. 基数が 16 なので、最初に値 512.1 を 16 進表記に変換する必要があります。
2. 最初に、基数 16 の記数法を使用して、整数部 512 を 16 進数に変換します。

基数 16						
16^7	...	16^4	16^3	16^2	16^1	16^0
268,435,456	...	65,536	4096	256	16	1

$$200 = .200 \times 16^3$$

値 512 は、16 進法で 200 として表現されます。

3. 16 進数 **200** を浮動小数点表現で記述します。これには、小数点を左端まで移動し、移動した桁数を数えます。移動した数が指数です。

$$200 = .200 \times 16^3$$

4. 元の数 512.1 の小数部(.1)を 16 進数に変換します。

$$.1 = \frac{1}{10} = \frac{1.6}{16}$$

分子は小数にできないので、1 を保持して.6 部分を再度変換します。

$$.6 = \frac{6}{10} = \frac{9.6}{16}$$

さらに、分子に小数を使用できないので、9 を保持して.6 部分を再変換します。

.6 は続けて 9.6 として繰り返します。つまり、9 を保持して再変換します。16 進法で表現できる.1 に最も近い値は、次のようになります。

$$.1 = .1999999 \times 16^0$$

5. 値の指数は 3 です(前述のステップ 2)。格納される実際の指数を決定するには、指数値を取り、そこにバイアスを足します。

```
true exponent + bias = 3 + 40 = 43 (hexadecimal) = stored exponent
```

最後に決定される部分は、仮数の符号です。慣習では、正の仮数の符号ビットは 0 で、負の仮数の符号は 1 です。この情報は、最初のバイトの最初のビットに格納されます。ステップ 4 の 16 進値から等値の 10 進数を計算し、バイナリ形式で書き込みます。符号ビットを最初の桁に追加します。格納された値は次のようになります。

$$43 \quad hexadecimal \quad = (4 \times 16^1) + (3 \times 16^0) = 67 \quad decimal \quad = 0100\,0003 \quad binary$$

$$11000003 = 195 \quad in \quad decimal \quad = C3 \quad in \quad hexadecimal$$

6. 最終ステップで、すべてをまとめます。

```
4320019999999999 - floating point representation for 512.1

C320019999999999 - floating point representation for -512.1
```

したがって、10 進値 512.1 は、2 進浮動小数点表現でも 16 進浮動小数点表現でも正確に表現できません。512.1 の数を変換すると、結果は無限に繰り返す数になります。これは、分数 1/3 を小数形式で表現するようなものです。

最近似値は、3 が無限に繰り返される **.33333333** です。

この例では、10 進表記で正確に表現できる数が、浮動小数点表現でも常に正確に表現できるとは限らないことを示しています。浮動小数点値に数の繰り返しパターン(前述の値における 9 の繰り返しなど)がある場合、値を正確に表現できない可能性が高いです。

精度誤差のトラブルシューティング

計算の留意点

精度の高さにかかわらず、一部の数値は正確に表現できません。ほとんどの有理数 (**.1** など)は、基数 2 や基数 16 では正確に表現できません。これは、浮動小数点表現で小数を格納するのは困難な場合が多いためです。

次の値の IBM メインフレーム表現について考えます。

```
1:  40 19 99 99 99 99 99 99
```

無限に繰り返す 9 の数字に注意してください。これは、1/3 を 10 進表現にしようとした場合 (**.3333** …) の末尾の 3 の数字に似ています。これらの値に対して算術演算を繰り返し実行すると、この精度不足が悪化する可能性があります。

たとえば、**33333** と **99999** を足すと、理論上の答えは **1.33333** になりますが、実際にはこの答えを出すことは不可能です。.値の計算が続くにつれて、合計が不正確になっていきます。

たとえば、次のような DATA ステップを実行するとします。

```
data _null_;
   do i=-1 to 1 by .1;
      put i=;
      if i=0 then put 'AT ZERO';
   end;
run;
```

DATA ステップ内の **AT ZERO** メッセージは出力されません。これは、不正確な数値の累積により、正確な値 0 が検出されないためです。計算結果は 0 に近くなりますが、正確に 0 に等しくなることはありません。したがって、浮動小数点で正確に数を表現できない場合、他の不正確な値で算術演算を実行することによって、より不正確になる可能性があります。

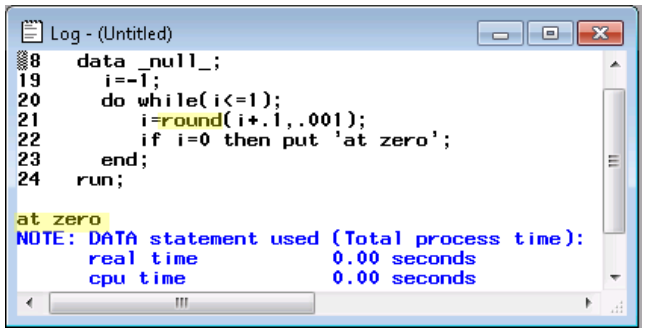
ROUND 関数による計算誤差の回避

不正確な値に対する計算実行の積み重ねが原因の誤差は、丸めによって解決できます。次の例では、ROUND 関数を使用して、結果を丸めたり、各反復について判断したりする方法を示します。

例のコード 4.7 *Using the ROUND Function to Avoid Computational Errors*

```
data _null_;
   do i=-1 to 1 by .1;
      i = round(i,.1);
      put i=;
```

```
      if i=0 then put 'AT ZERO';
   end;
run;
```



もう 1 つの例では、z/OS では発生するが PC では発生しない数値精度の問題を示します。

例のコード 4.8 *Using the ROUND Function with the IF Statement*

```
data a;
   input gender $ height;
datalines;
  m 60
  m 58
  m 59
  m 70
  m 60
  m 58 ;
run;

proc freq;
tables gender/out=new;
run;

data final;
   set new;
   if percent=100 then put 'equal';
   else put 'not equal';
run;
```

The FREQ Procedure

gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
m	6	100.00	6	100.00

例では、PROC FREQ で、変数 Percent を含む出力データセットが作成されます。変数 Gender の値がすべて同じなので、Percent は正確な値の 100 になると考えられます。ところが、Percent の値がテストされると、ログで、Percent が正確に 100 ではないことが示されます。

PROC FREQ で変数 Percent を生成するために使用されるアルゴリズムには、数学的計算が含まれます。結果は 100 に非常に近くなりますが、正確に 100 にはなりませ

ん。IF ステートメントで ROUND 関数(または COMPFUZZ 関数)を使用すると、この問題が解決します。

非常に単純な計算(たとえば、小数点第 2 位までしかないなど)の回避策は、値に 100 を掛けて、ROUND 関数を使用して整数に丸めることです。いったん新しい整数に対して計算を実行したら、100 で割り、値を元の小数形式に変換します。

次の例では、変数 **x** の値は SAS データセットに実数として格納されます。その数に 1,000 を掛け、ROUND 関数を使用して値を整数に変更します。SUM ステートメントを使用して、**New** のすべての値を合計します。最後のオブザベーション(END=オプションを使用して検出する)で、合計を 1,000 で割り、値を元の分数に変換します。

例のコード 4.9 *Summing Rounded Values*

```
data a;
set b end=last;
new=round(x*1000);
sum+new;
if last then sum=sum/1000;
run;
```

この関数の詳細については、“ROUND Function” (*SAS Functions and CALL Routines: Reference*)を参照してください。

数値の比較の留意点

正確な 10 進または 16 進浮動小数点表現を持たない非正数値の比較時に、驚くべき結果に遭遇することがあります。たとえば、10 進演算では、式

```
15.7 - 11.9 = 3.8
```

は真になります。ところが、SAS では、リテラル値 **3.8** と計算値 **15.7 - 11.9** を比較し、その結果を SAS ログに出力すると、'not equal'という結果になります。

例のコード 4.10 *Comparing Values That Have Imprecise Representations*

```
data a;
  x=15.7-11.9;
  if x=3.8 then put 'equal';
    else put 'not equal';
run;
```

ログ 4.2 *不正確な表現の値の比較についてのログ出力*

```
988     data a; 989     x=15.7-11.9; 990     if x=3.8 then put 'equal';; 991
else put 'not equal'; 992     run; not equal NOTE:The data set WORK.A has 1
observations and 1 variables.NOTE:DATA statement used (Total process time): real
time  0.00 seconds cpu time   0.01 seconds
```

ログ出力は、値 **3.8** と **(15.7 - 11.9)** が同値ではないことを示します。これは、計算にかかわる値を、2 進法でも 16 進法でも正確に表現できないためです。

PRINT プロシジャを追加して結果を表示すると、PROC PRINT 出力が格納値と異なることを確認できます。PROC PRINT ステートメントでは、**x** の値は、実際の格納値ではなく **3.8** として表示されます。これは、プロシジャが、自動的に出力形式を適用し、結果を丸めてから表示するためです。その場合、PROC PRINT は、計算後の最終結果のみを丸めます。次の例では、それが原因で、非明示的丸めによって混乱が引き起こされる可能性があることを示します。

```
proc print data=a;
run;
```

アウトプット 4.6 *値比較の出力*

The SAS System

Obs	x
1	3.8

出力形式による精度誤差の確定

次の例では、例のコード 4.10 (70 ページ) で得られた結果に 2 つの異なる出力形式が適用され、SAS ログに表示されます。最初の出力形式 `10.8` は、x の値が `3.8` であることを示しますが、10.16 出力形式を使用した値の表示は、x が `3.8` よりわずかに小さいことを示します。

例のコード 4.11 *Using Formats to Confirm Precision Errors*

```
data a;
x=15.7-11.9;
if x=3.8 then put 'equal';
   else put 'not equal';
put x=10.8;
put x=18.16;
run;
```

ログ 4.3 *ログ出力:出力形式による精度誤差の確定*

```
102  data a; 103    x=15.7-11.9; 104    if x=3.8 then put 'equal'; 105      else
put 'not equal'; 106  put x=10.8; 107  put x=18.16; run; not equal x=3.80000000
x=3.7999999999999900 NOTE:The data set WORK.A has 1 observations and 1
variables.NOTE:DATA statement used (Total process time): real time 0.01 seconds
cpu time 0.01 seconds
```

x の格納値を確認するもう 1 つの方法は、HEX16.出力形式を計算結果に適用することです。HEX16.出力形式は、浮動小数点表現の表示に使用できる特別な出力形式です。

例のコード 4.12 *Using the HEX16 Format to Verify Calculated Results*

```
data a;
  x=15.7-11.9;
  if x=3.8 then put 'equal';
    else put 'not equal';
put x=hex16.;
run;
```

ログ 4.4 *HEX16 出力形式による計算結果の確認*

```
123  data a; 124    x=15.7-11.9; 125    if x=3.8 then put 'equal'; 126      else
put 'not equal'; 127  put x=hex16.; 128  run; not equal x=400E666666666664
NOTE:The data set WORK.A has 1 observations and 1 variables.NOTE:DATA statement
used (Total process time): real time 0.03 seconds cpu time 0.03 seconds
```

この出力形式の詳細については、"HEXw. Format" (*SAS Formats and Informats: Reference*)を参照してください。出力形式全般の詳細については、"Dictionary of Formats" (*SAS Formats and Informats: Reference*)を参照してください。

ROUND 関数による比較誤差の回避

比較の実行前に明示的に値を丸めることによって比較誤差を回避できます。次の例では、1/3 の計算結果と割り当て値 .33333 を比較します。1/3 が不正確な数なので、値は .33333 に等しくならず、PUT ステートメントは実行されません。

例のコード 4.13 *Using the ROUND Function to Avoid Comparison Errors*

```
data _null_;
   x=1/3;
   if x=.33333 then put 'MATCH';
run;
```

ただし、次の例のように ROUND 関数を追加すると、PUT 'MATCH'ステートメントが実行されます。

```
data _null_;
   x=1/3;
   if round(x,.00001)=.33333 then put 'MATCH';
run;
```

ログ 4.5 *ログ出力:ROUND 関数による比較誤差の回避*

```
NOTE:SAS initialization used: real time           1.68 seconds cpu
time            0.98 seconds 1    data _null_; 2       x=1/3; 3       if
round(x,.00001)=.33333 then put 'MATCH'; 4    run; MATCH NOTE:DATA statement
used (Total process time): real time           0.02 seconds cpu time
0.01 seconds
```

通常、小数値(分数値)の比較を行う場合は、計算や比較を実行する前に ROUND 関数を使用することをお勧めします。

この関数の詳細については、"ROUND Function" (*SAS Functions and CALL Routines: Reference*)を参照してください。

値比較時の LENGTH ステートメントの使用

LENGTH ステートメントを使用すると、変数値を格納するために使用するバイト数を制御できます。ただし、誤差や重大なデータ損失を回避するには注意深く使用する必要があります。

たとえば、IBM メインフレーム表現では、完全な精度に 8 バイトを使用しますが、2 バイトだけでディスク領域に格納できます。この場合、値 1 は次のように 8 バイトで示されます。

```
41 10 00 00 00 00 00 00
```

2 バイトの場合は、41 10 に切り捨てられます。この場合、それでも指数が完全な状態のままなので大きさは完全に保たれますが、桁数は減ります。桁数が減るということは、末尾の 0 の前の小数部の右または左の桁数が少なくなることを意味します。

たとえば、1234567890 という数について考えます。これは、基数 10 の浮動小数点表現で .1234567890 に 10 の 10 乗を掛けたものです。.5 桁の精度しかない場合、この数は 123460000 になります(切り上げ)。これは、使用される 10 の累乗に関係なくあてはまることに注意してください(.12346, 12.346, .0000012346 など)。

また、前述のように、慎重に長さを指定する必要があります。IBM メインフレームシステム上での 2 バイトの長さについて考えます。この場合、指数と符号を格納するために 1 バイトを使用でき、指数のために 1 バイトを使用できます。1 バイトに格納できる最大値は、255 です。したがって、指数が 0 の場合(16 の 0 乗、つまり 1 を仮数に掛

ける場合)、完全な精度で格納できる最大の整数は 255 です。ただし、16 の倍数である場合は、これより大きな整数でも格納できます。

たとえば、256 から 272 までの数の 8 バイト表現について、次の表で考えます。

表 4.10 *256 から 272 までの数の 8 バイト表現の表*

値	符号/指数	仮数 1	仮数 2 - 7	留意点
256	43	10	000000000000	16 の倍数で末尾は 0
257	43	10	100000000000	さらにバイトが必要
258	43	10	200000000000	
259	43	10	300000000000	
...	...	...	...	...
271	43	10	F00000000000	
272	43	11	000000000000	16 の倍数で末尾は 0

257 から 271 までの数は、最初の 2 バイトでは正確に格納できません。この数を正確に格納するには、3 番目のバイトが必要です。したがって、次のコードでは、正しい結果は得られません。

```
data temp;
  length x 2;
  x=257;
  y1=x+1;
run;

data _null_;
set temp;
if x=257 then put 'FOUND';
y2=x+1;
run;
```

実際は X の値が 256 なので(値 257 は 2 バイトに切り捨てられる)、PUT ステートメントは実行されません。256 は 4310 として 2 バイトに格納されますが、257 も 4310 として 2 バイトに格納されます。257 では、3 番目のバイトの 10 が切り捨てられます。

最初の DATA ステップで値 257 が切り捨てられるという警告は表示されません。ただし、Y1 の値は 258 であることに注意してください。これは、X の値が、完全な 8 バイトの浮動小数点表現でプログラムデータベクトルに保存されるためです。値は、SAS データセットに格納される場合のみ切り捨てられます。Y2 の値は 257 です。これは、X が切り捨てられてから、その数がプログラムデータベクトルに読み込まれるためです。

注意:

変数値が整数でない場合は、LENGTH ステートメントを使用しないでください。 整数でない数値を切り捨てると、精度が失われます。また、ディスク領域が十分でない場合だけ、LENGTH ステートメントを使用して値を切り捨てます。最大値は、使用している動作環境に対応する SAS ドキュメントを参照してください。

このステートメントの詳細については、“LENGTH Statement” (*SAS Statements: Reference*)を参照してください。

値比較時の TRUNC 関数の使用

TRUNC 関数は、数値を要求された長さに切り捨ててから、その数値を完全な長さに展開します。切り捨てとその後の展開は、完全な長さより短く数値を格納してからその数値を読み込む場合と同じ影響を及ぼします。たとえば、変数

```
x = 1/3
```

という変数を長さ 3 で格納すると、次の比較は真にはなりません。

```
if x = 1/3 then ...;
```

ただし、次のように TRUNC 関数を追加すると、この比較は真になります。

```
if x=trunc(1/3,3) then ...;
```

この関数の詳細については、“TRUNC Function” (*SAS Functions and CALL Routines: Reference*)を参照してください。

数の正確な格納に必要なバイト数の決定

TRUNC 関数を使用すると、値を正確に格納するために必要な最小バイト数を決定できます。次のプログラムは、IBM メインフレーム環境で Numbers という名前のネイティブ SAS データセットに格納される数に必要な最小バイト長(MinLen)を割り出します。データセット Numbers には、変数 Value が含まれます。Value には、**269** から **272** までの範囲の数が含まれます。

例のコード 4.14 *Determining How Many Bytes Are Needed to Store a Number Accurately*

```
data numbers;
input value;
datalines;
269
270
271
272
;

data temp;
   set numbers;
    x=value;
   do L=8 to 1 by -1;
   if x NE trunc(x,L) then
      do;
         minlen=L+1;
         output;
         return;
      end;
   end;
run;

proc print noobs;
   var value minlen;
 run;
```

次の出力は、この例の結果を示しています。

アウトプット 4.7 数の正確な格納に必要なバイト数の決定

The SAS System

value	minlen
269	3
270	3
271	3
272	2

値 271 に必要な最小長が、値 272 に必要な最小長より大きいことに注意してください。これは、ある範囲内の数値で、最大値が、それより小さな数値より少ないバイト数で格納できる場合があることを示します。範囲内のすべての数値に高い精度が必要な場合は、最大値だけではなく、すべての数値に必要な最小長を取得する必要があります。

この関数の詳細については、“TRUNC Function” (*SAS Functions and CALL Routines: Reference*)を参照してください。

倍精度と単精度浮動小数点数

外部プログラムによって作成されたデータを SAS データセットに読み込みたい場合があります。データが浮動小数点表現の場合は、RBw.d 入力形式を使用して、データを読み込むことができます。ただし、例外があります。**RBw.d** 入力形式は、w 値が倍精度浮動小数点数のサイズ(このセクションで取り上げたすべてのオペレーティングシステムで 8)より小さい場合、倍精度浮動小数点数を切り捨てることがあります。したがって、**RB8.**入力形式は、完全な 8 バイトの浮動小数点に対応します。**RB4.**入力形式は、4 バイトに切り捨てられた 8 バイトの浮動小数点に対応し、DATA ステップ内の LENGTH 4 とまったく同じです。

4 バイトに切り捨てられた 8 バイトの浮動小数点は、C プログラムの浮動小数点とは同じにならないことがあります。8 バイトの浮動小数点数は、C 言語では double、FORTRAN では REAL*8、IBM の PL/I では FLOAT BINARY(53)と呼ばれます。4 バイトの浮動小数点数は、C 言語では float、FORTRAN では REAL*4、IBM の PL/I では FLOAT BINARY(21)と呼ばれます。

IBM メインフレームの場合、単精度浮動小数点数は、4 バイトに切り捨てられた倍精度浮動小数点数とまったく同じです。IEEE 規格を使用するオペレーティングシステムでは、これが当てはまりません。つまり、単精度浮動小数点数が、指数に異なるビット数を使用し、異なるバイアスを使用するので、RB4.入力形式を使用して値を読み込むと、期待した結果が得られません。

オペレーティングシステム間でのデータの転送

浮動小数点表示で表現された非常に大きな数値や非常に小さな数値を含むデータを転送すると、精度や大きさの問題が発生する可能性があります。表 4.9 (62 ページ)は、基数、指数、仮数の最大桁数を示しています。オペレーティングシステムが異なる

と、格納できる最大値も異なるため、浮動小数点データをあるコンピュータから別のコンピュータに移送する場合に問題が生じることがあります。

たとえば、IBM メインフレームと PC の間での移送について考えます。IBM メインフレームには、およそ.**54E-78** から.**72E76** まで(およびその負の数字と 0)の範囲制限があります。

PC などのその他のコンピュータの制限は、IBM メインフレームより広範です(PC の上限はおよそ 1E308)。したがって、1E100 の大きさの数値を PC からメインフレームに移送する場合は、その大きさが失われます。制限を超える数値は、データ転送中に、そのオペレーティングシステムで許可されている最小値または最大値に設定されます。したがって、PC 上の 1E100 は、IBM メインフレーム上ではおよそ.72E76 という値に変換されます。

注意:
コンピュータ間でのデータの転送は、数値精度に影響を及ぼすことがあります。

IBM メインフレームから PC にデータを転送する場合は、仮数のビット数が IBM メインフレームよりも 4 つ少ないことに注意してください。これは、PC に移送するときに 4 ビットを失うことを意味します。

このような数値の精度と大きさの違いは、ある動作環境から、浮動小数点表現の異なる別の動作環境にデータを移送する場合に考慮すべき事柄です。

代替案として、また、おそらくはオペレーティングシステム間のデータ転送時の数値精度問題の最も安全な回避策としては、データ内の数を整数に変換する方法があります。

オペレーティングシステム間のデータ移動の詳細については、*SAS ファイルの移動とアクセス*を参照してください。

5 章

欠損値

欠損値の定義

欠損値

欠損値は、現在のオブザベーションの変数に、データ値が格納されていないことを示す値です。欠損値には次の 3 種類があります。

- 数値
- 文字
- 特殊数値

デフォルトでは、数値欠損値は 1 個のピリオド(.)で出力されます。また、文字欠損値は 1 個のブランクで出力されます。特殊数値欠損値の詳細については、“特殊欠損値の作成” (78 ページ) を参照してください。

特殊欠損値の作成

定義

特殊欠損値
数値欠損値の一種です。特殊欠損値を使用すると、各種の欠損データをアルファベット(A - Z)とアンダースコア(_)を使用して表すことができます。

ヒント

- SAS では、大文字、小文字のどちらでも使用できます。値の表示と出力は大文字で行われます。
- 特殊数値欠損値の先頭にピリオド(.)が付いていない場合、SAS System では変数名として認識します。このため、SAS 式または割り当てステートメントの中で特殊数値欠損値を使用するときは、値の先頭にピリオド(.)を付け、その後にアルファベットまたはアンダースコア(_)を入力してください。次にその例を示します。

```
x=.d;
```

- 特殊欠損値の出力時には、アルファベットまたはアンダースコア(_)だけが出力されます。
- データ値の数値フィールドに文字が含まれている場合に、その文字が特殊欠損値として解釈されるようにするには、MISSING ステートメントを使用して必要な文字を指定します。詳細については、“MISSING Statement” (*SAS Statements: Reference*)を参照してください。

例

例として、市場調査会社のデータを使用します。ここに 5 種類の製品があります。5 人のテスターが雇われ、製品の使いやすさと効果を 1 種類ずつテストすることになりました。テスターが休んだ場合は、評価レポートがないので、値は“欠”を表す X として記録されます。テスターが製品を十分にテストできなかった場合は、評価レポートがないので、値は“テスト不完全”を表す I として記録されます。次のプログラムは、データを読み込んで、処理結果の SAS データセットを表示するものです。1 番目と 3 番目のデータ行にある特殊欠損値に注目してください。

```
data period_a;
  missing X I;
  input Id $4. Foodpr1 Foodpr2 Foodpr3 Coffeem1 Coffeem2;
  datalines;
1001 115 45 65 I 78
1002 86 27 55 72 86
1004 93 52 X 76 88
1015 73 35 43 112 108
1027 101 127 39 76 79
  ;

proc print data=period_a;
  title 'Results of Test Period A';
  footnote1 'X indicates TESTER ABSENT';
```

```
    footnote2 'I indicates TEST WAS INCOMPLETE';
run;
```

出力結果は次のようになります。

アウトプット 5.1 *複数の欠損値が含まれる出力結果*

Results of Test Period A

Obs	Id	Foodpr1	Foodpr2	Foodpr3	Coffeem1	Coffeem2
1	1001	115	45	65	I	78
2	1002	86	27	55	72	86
3	1004	93	52	X	76	88
4	1015	73	35	43	112	108
5	1027	101	127	39	76	79

X indicates TESTER ABSENT
I indicates TEST WAS INCOMPLETE

欠損値の順序

数値変数

SAS System では、数値変数の欠損値は他のすべての数値よりも小さいものと見なされます。数値変数を基準としてデータセットを並べ替えた場合、欠損値が格納されているオブザベーションは、並べ替えられたデータセットの先頭に表示されます。数値変数では、特殊欠損値を数値と比較することや、特殊欠損値どうしを比較することができます。次の表は、数値の並べ替え順序を示しています。

表 5.1 *数値の並べ替え順序*

並べ替え順序	記号	説明
最小	._	アンダースコア
	.	ピリオドの数値欠損値
	.A - Z	A (最小) - Z (最大)の特殊欠損値
	-n	負の数値
	0	ゼロ
最大	+n	正の数値

たとえば、数値欠損値(.)は、特殊数値欠損値である.A の前に出力されます。また、. も.A も特殊欠損値.Z の前に出力されます。特殊数値欠損値の並べ替えでは、アルファベットの大文字と小文字は区別されません。

注: 数値欠損値の並べ替え順序は、ASCII 照合順序と EBCDIC 照合順序のどちらが使用されていても変わりません。

文字変数

文字変数の欠損値は、他のすべての表示可能な文字値よりも小さいものと見なされます。したがって、文字変数を基準としてデータセットを並べ替えると、BY 変数が欠損値(ブランク)になっているオブザベーションは、BY 変数の値に表示可能な文字だけが格納されているオブザベーションよりも前に表示されます。ただし、表示不能な文字の中には、通常はブランクより小さい値を持つものがあります。コンピュータのキャリッジコントロール文字や、誤って文字データとして読み取られたバイナリ実数データなどです。そのため、表示不能な文字がデータ内にある場合は、並べ替えられたデータセットの先頭に欠損値が出力されないことがあります。

SAS により変数値が欠損値に自動設定される場合

生データを読み込む場合

DATA ステップで作成される変数の値は、DATA ステップの反復が開始される前に、自動的に欠損値に設定されます。ただし、次のような変数は例外です。

- RETAIN ステートメント内に指定されている変数
- SUM ステートメントで作成された変数
- _TEMPORARY_配列のデータ要素
- FILE ステートメントまたは INFILE ステートメントで、オプションを使用して作成された変数
- FGET 関数で作成された変数
- ARRAY ステートメント内で初期化されているデータ要素
- 自動変数

SAS System では、変数に値が割り当てられたことを検出した時点で、欠損値を自動的に置き換えます。このため、プログラムステートメントを使用して新しい変数を作成した場合は、割り当てステートメントで値を割り当てるまで、各オブザベーションの変数の値は欠損値のままです。

```
data new;
      input x;
      if x=1 then y=2;
      datalines;
   4
   1
   3
   1
   ;
```

DATA ステップの例を次に示します。この DATA ステップでは、次の変数値を持つ SAS データセットが生成されます。

```
OBS   X   Y
  1   4   .
  2   1   2
  3   3   .
  4   1   2
```

X が 1 のとき、Y の値は 2 に設定されます。X が 1 以外の場合には、Y の値を設定するステートメントが他にないので、そのオブザベーションの Y は欠損値(.)のまま残ります。

SAS データセットの読み込み時

SET ステートメント、MERGE ステートメント、UPDATE ステートメントによって変数が読み込まれるときは、DATA ステップの最初の反復の前に限り、値が欠損値に設定されます。(BY ステートメントを使用した場合、変数値は BY グループが変化するときも欠損値に設定されます)。変数の値は、(割り当てステートメント、SET ステートメント、MERGE ステートメント、UPDATE ステートメントなどを介して)新しい値が割り当てられるまで維持されます。SET ステートメント、MERGE ステートメント、UPDATE ステートメントで、オプションを使用して作成した変数の値も次の反復まで維持されます。

BY ステートメントを使用したマッチマージ操作では、データセットの行がすべて処理されたときに、出力データセット内の変数の値が前述のように維持されます。つまり、データセットの行がすべて処理されても、BY 変数の値に有効な変化がない場合には、出力データセットの変数では直前のオブザベーションの値が維持されます。BY ステートメントの処理では、FIRST.variable と LAST.variable という自動変数が生成されますが、これらの変数の値はいずれも維持されます。これらの自動変数の初期値は 1 です。

BY 変数の値が変化すると、これらの変数は欠損値に設定され、欠損値のまま維持されます。これは、データセットの中に、置き換えのための値を提供するオブザベーションが他に含まれていないためです。BY ステートメントを使用しない 1 対 1 のマージでは、データセットの行がすべて処理されたときに、出力データセット内の変数の値が欠損値に設定され、欠損値のまま維持されます。

SAS により欠損値が生成される場合

計算における欠損値のプロパゲーション

SAS System では、問題の発生を防ぐために欠損値が割り当てられることがあります。算術計算で欠損値を使用すると、その計算結果は自動的に欠損値に設定されます。その計算結果を他の計算で使用すると、次の計算結果も欠損値になります。この動作を欠損値のプロパゲーションと呼びます。SAS ログには、欠損値が含まれる算術演算式と、その作成日時を示すメッセージが出力されます。ただし、処理は継続されます。

無効な演算

次のような無効な演算を実行しようとすると、SAS ログに NOTE メッセージが出力され、演算結果には欠損値が割り当てられます。

- 0 による除算

- 0 の対数計算
- 計算結果が浮動小数点数で表現できないほど大きな数値になる式(オーバーフロー)

無効な文字から数値への変換

算術演算式で文字変数を使用した場合、文字値は自動的に数値に変換されます。このとき、文字値に数以外の情報が含まれていると、SAS System はそれを数値に変換しようとし、SAS ログにはメッセージが表示されます。変換結果は欠損値になり、自動変数_ERROR_の値が 1 に設定されます。

特殊欠損値の作成

SAS 式に数値欠損値があると、計算結果はすべてピリオドになります。.このため、特殊欠損値と通常の数値欠損値は、どちらもピリオドとしてプロパゲートします。

```
data a;
      x=.d;
      y=x+1;
      put y=;
   run;
```

この DATA ステップを実行すると、次のような SAS ログが表示されます。

ログ 5.1 *欠損値を示す SAS ログ*

```
130  data a; 131        x=.d; 132        y=x+1; 133        put y=; 134     run;
y=.NOTE:Missing values were generated as a result of performing an operation on
missing values.Each place is given by:(Number of times) at (Line):(Column).1 at
132:10 NOTE:The data set WORK.A has 1 observations and 2 variables.NOTE:DATA
statement used (Total process time): real time 0.00 seconds cpu time 0.00 seconds
```

欠損値のプロパゲーションの防止

算術演算式に欠損値がプロパゲートしないようにするには、記述統計関数を使用して欠損値を計算の対象から除外します。これらの関数のリストについては、“SAS Functions and CALL Routines by Category” (*SAS Functions and CALL Routines: Reference*)を参照してください。たとえば、次のような DATA ステップを実行するとします。

```
data test;
  x=.;
  y=5;
  a=x+y;
  b=sum(x,y);
  c=5;
  c+x;
  put a= b= c=;
run;
```

ログ 5.2 統計関数内の欠損値を示す SAS ログ

```
143 data test; 144 x=.; 145 y=5; 146 a=x+y; 147 b=sum(x,y); 148 c=5; 149 c+x;
150 put a= b= c=; 151 run; a=. b=5 c=5 NOTE:Missing values were generated as a
result of performing an operation on missing values.Each place is given by:
(Number of times) at (Line):(Column).1 at 146:6 NOTE:The data set WORK.TEST has
1 observations and 5 variables.NOTE:DATA statement used (Total process time):
real time 0.12 seconds cpu time 0.01 seconds
```

X の値が欠損値のため、式を使用して X と Y を加算すると、結果は欠損値になります。ここでは、A の値が欠損値になっています。ただし、SUM 関数では欠損値が無視されるので、X と Y を加算したときの値は欠損値ではなく 5 になります。

注: SUM ステートメントでも欠損値は無視されるので、C の値も 5 になります。

欠損値の処理

生データにおける欠損値の表現方法

次の表は、生データに含まれる欠損値の表現方法を、欠損値の種類ごとに示しています。この表に従って欠損値を記述することで、値の読み込みと格納が適切に行われます。

表 5.2 欠損値の表示

欠損値	データの表現
数値	.(1 個のピリオド)
文字	' ' (引用符で囲まれたブランク)
特殊	.文字(1 個のピリオドとそれに続く 1 文字のアルファベット。例:.B)
特殊	._(1 個のピリオドとそれに続くアンダースコア)

DATA ステップで変数値を欠損値に設定する方法

DATA ステップ内で値を欠損値に設定するには、次のようなプログラムステートメントを使用します。

```
if age<0 then
age=.;
```

このステートメントでは、格納されている変数 Age の値が 0 未満の場合に、Age の値が数値欠損値に設定されます。

注: DATA ステップ内で MISSING ステートメントまたは MISSING=システムオプションを使用すると、数値欠損値をピリオド(.)以外の文字で表示することができます。

次の例では、変数 Name の値が“none”である場合に、Name の値が文字欠損値に設定されます。

```
if name="none" then name='';
```

または、1 つ以上の変数値の欠損値に設定する場合は、CALL MISSING ルーチンを使用できます。たとえば、次のようになります。

```
call missing(sales, name);
```

これにより、両方の変数値が欠損値に設定されます。

注: CALL MISSING ルーチンの引数リストで文字変数と数値変数を混在させることができます。

DATA ステップにおける欠損値のチェック方法

N 関数と NMISS 関数を使用すると、数値引数のリストから非欠損値の数と欠損値の数を返すことができます。

通常の欠損数値をチェックする場合、次のようなコードを使用します。

```
if numvar=. then do;
```

データに特殊欠損値が含まれる場合、次のようなコードを使用して、通常欠損値または特殊欠損値をチェックできます。

```
if numvar<=.z then do;
```

欠損文字値をチェックするには、次のようなステートメントを使用します。

```
if charvar=' ' then do;
```

MISSING 関数を使用すると、次のように文字欠損値または数値欠損値をチェックできます。

```
if missing(var) then do;
```

いずれの場合でも、SAS は現在のオブザベーション内にある変数の値が、指定された条件を満たすかどうかをチェックします。条件を満たしている場合は、DO グループが実行されます。

注: 欠損値を AND や OR などの論理演算子とともに使用すると、評価結果は **false** になります。

6 章

SAS 式

SAS 式の定義

式
: 値を算出するための命令をオペランドと演算子で構成したものを、SAS 式と呼びます。SAS System のプログラムステートメントでは、SAS 式を使用することで、変数の作成、値の割り当て、新しい値の計算、変数の変換、条件付き処理を行います。SAS 式は、展開された結果、数値、文字値、0 か 1 のブール値を生成して、新しい値を算出します。

オペランド
: SAS 式の中で演算処理対象とする、定数または変数を指定します。数値または文字のいずれも指定できます。

演算子
: 比較演算、算術演算、論理演算を表す記号です。SAS 関数や、グループ化のためのかっこも演算子の一種です。

単純式
: 演算子が 1 つだけ含まれる式のことです。単純式は、次の単一の演算子のいずれか 1 つで構成されます。

  - 定数
  - 変数
  - 関数

複合式
: 複数の演算子が含まれる式のことです。複合式が検出されると、複合式の評価規則に従って式の各部分の評価順序が決定されます。

WHERE 式
: SAS 式の一種で、WHERE ステートメントまたは WHERE=データセットオプションの中で使用されます。WHERE 式には、DATA ステップまたは PROC ステップで処理するオブザベーションを選択するための条件を指定します。WHERE 式の構文や詳細については、11 章, “WHERE 式の処理” (165 ページ)を参照してください。

SAS 式の例

SAS 式の例を次に示します。

- `3`
- `x`
- `x+1`
- `age<100`
- `trim(last)||', '||first`

式内の SAS 定数

定義

SAS 定数は、値が固定されている数値または文字列です。SAS 定数は、変数割り当てステートメントや IFTHEN ステートメントなど、多くの SAS ステートメントで式として使用できます。また、特定のオプションの値としても使用できます。定数はリテラルとも呼ばれます。

SAS 定数には、次の種類があります。

- 文字
- 数値
- 日付、時間、日時
- ビットテスト

文字定数

文字定数は、1 - 32,767 個の文字で構成されます。文字定数を使用するときは、引用符で囲む必要があります。文字定数は 16 進表記で表現することもできます。

引用符と文字定数の併用

次の SAS ステートメントでは、**Tom** が文字定数です。

```
if name='Tom' then do;
```

文字定数に一重引用符(')が含まれている場合は、二重引用符(")で囲む必要があります。たとえば、**Tom's** という文字値を定数として指定するには、次のように入力します。

```
name="Tom's"
```

もう 1 つの方法としては、文字列を一重引用符(')で囲み、アポストロフィを表記するために一重引用符を 2 つ続けて記述します。一重引用符を 2 つ続けて記述すると、1 つの一重引用符として扱われます。

```
name='Tom''s'
```

この規則は二重引用符の場合にも当てはまります。

```
name="Tom""s"
```

注意:

引用符の始まりと終わりの対応を適正にする必要があります。 引用符の数が多いあるいは少ない場合には、誤りのあるステートメントと後続するステートメントは、どちらも誤って解釈されます。たとえば、**name='O'Brien';**のように指定した場合、**O** は変数 Name の文字値、**Brien** は無意味なトークン、**';**は引用符で囲まれた次の文字列の始まりとして扱われます。

文字定数と文字変数の比較

文字定数は引用符で囲みますが、文字変数名は引用符で囲みません。文字定数と文字変数を混同しないようにしてください。この違いは、文字定数を使用するすべての場合に適用されるため、重要です。たとえば、タイトル、フットノート、ラベル、オプション値、動作環境固有のファイル指定やコマンドなどで、文字定数を使用する場合は引用符で囲むことが必要です。

次のステートメントでは、文字定数を使用しています。

- **x='abc';**
- **if name='Smith' then do;**

次のステートメントでは、文字変数を使用しています。

- **x=abc;**
- **if name=Smith then do;**

文字変数の例では、定数ではなく、ABC、SMITH という名前の変数がそれぞれ検索されます。

注: 文字式を比較するときは、アルファベットの大文字と小文字が区別されます。たとえば、文字値**'Smith'**と**'SMITH'**は等しくありません。

16 進表記で表現された文字定数

SAS System の文字定数は、16 進表記で表現できます。16 進数文字定数は、偶数個の 16 進文字で構成される文字列です。次の例に示すように、一重引用符(')または二重引用符(")で囲み、直後に X を付けます。

```
'534153'x
```

文字列を読みやすくするために、カンマ(,)を使用してもかまいません。ただし、カンマを 16 進値の一部にしたり、カンマによって 16 進値を変更したりすることはできません。文字列にカンマを含める場合、カンマは文字列の中で偶数個ごとに 16 進文字を区切るものでなければなりません。次に例を示します。

```
if value='3132,3334'x then do;
```

注: 引用符内に後置ブランクまたは前置ブランクがあると、エラーメッセージがログに書き出されます。

数値定数

数値定数とは、SAS ステートメント内で使用される固定的な数値です。数値定数は、次の形式で表記することができます。

- 標準表記
- 指数表記
- 16 進表記

標準表記で表現された数値定数

ほとんどの数値定数は、数値データと同じように記述できます。次の式では、数値定数は 100 です。

```
part/all*100
```

数値定数は標準表記で表すことができます。次に例を示します。

表 6.1 *数値定数の標準表記*

数値定数	説明
1	符号なし整数です。
-5	負符号(-)が付いています。
+49	正符号(+)が付いています。
1.23	小数点(.)が付いています。
01	重要ではない前置ゼロが付いています。

指数表記で表現された数値定数

指数表記では、E の前にある数字(仮数部)に、E の後にある数字(指数部)に対応する 10 の累乗を掛けることで特定の数を表します。たとえば、2E4 は $2x10^4$、すなわち 20,000 です。$(10^{32})-1$ より大きな数値定数の場合は、指数表記を使用する必要があります。次に例を示します。

- **`1.2e23`**
- **`0.5e-10`**

16 進表記で表現された数値定数

16 進数値として表記される数値定数は、数字(通常は 0)で始まり、さらに 16 進数字が続き、最後に文字 X が付きます。数値定数には、16 桁までの有効な 16 進表記の数字(0 - 9、A - F)を格納できます。次に 16 進数値定数の例を示します。

- **`0c1x`**
- **`9x`**

DATA ステップでは、次のように 16 進数値定数を使用できます。

```
data test;
   input abend pib2.;
   if abend=0c1x or abend=0b0ax then do;
     more SAS statements
run;
```

日付定数、時間定数、日時定数

日付定数、時間定数、日時定数を作成するには、日付または時間を一重引用符(')または二重引用符(")で囲み、後ろに D (日付)、T (時間)、DT (日時)を付けて値の種類を示します。

引用符内に後置ブランクまたは前置ブランクがあっても、日付定数、時間定数、日時定数の処理には影響しません。

日時定数を作成する場合は、次のパターンを使用します。

'*ddmmm<yy>yy*'D または“*ddmmm<yy>yy*”D は SAS 日付値を表します。

- `date='1jan2013'd;`
- `date='01jan09'd;`

'*hh:mm<:ss.s>*'T または“*hh:mm<:ss.s>*”T は SAS 時間値を表します。

- `time='9:25't;`
- `time='9:25:19pm't;`

'*ddmmm<yy>yy:hh:mm<:ss.s>*'DT または“*ddmmm<yy>yy:hh:mm<:ss.s>*”DT は SAS 日時値を表します。

- ```
  if begin='01may12:9:30:00'dt then
  end='31dec13:5:00:00'dt;
  ```
- `dtime='18jan2003:9:27:05am'dt;`

'*yyyy-mm-dd*T*hh:mm:ss*Z'DT または'*yyyy-mm-dd*T*hh:mm:ss*+|-*hh:ss*'DT は、ISO 8601 標準に基づく UTC(協定世界時)の SAS 日時定数を表します。

- `tstamp='2013-05-17T09:15:30-05:00'dt;`および `tstamp='2013-05-17T09:15:30-05'dt;`は、米国東部標準時の UTC を表します。
- `tstamp='2013-07-20T12:00:00+00:00'dt;`および `tstamp='2013-07-20T12:00:00Z'dt;`は、英国グリニッジ近くの基準子午線の UTC を表します。

SAS 日付の詳細については、7 章, “日付、時間と間隔” (105 ページ)を参照してください。

ビットテスト定数

ビットマスクを使用することで、データ値の表記における内部ビットを比較するビットテストを実行できます。ビットテストは、文字変数と数値変数のどちらに対しても行えます。ビットテスト演算は、次の構文で指定します。

expression comparison-operator bit-mask

各構成要素は、次のとおりです。

expression (式)
: 任意の有効な SAS 式を指定できます。文字変数と数値変数の両方をビットテストの対象にすることができます。文字値をテストする場合は、ビットマスクの左端ビットと文字列の左端ビットが揃えられ、右へ向かって、対応するビット同士がテストされます。数値をテストする場合、浮動小数点数は 32 ビットの整数へと切り捨てられます。ビットマスクの右端ビットと数字の右端ビットが揃えられ、左へ向かって、対応するビット同士がテストされます。

comparison-operator (比較演算子)
: 式とビットマスクを比較します。これらの演算子の説明については、“比較演算子” (94 ページ)を参照してください。

bitmask (ビットマスク)
: 複数の 0、1、ピリオド(.)から成る、引用符で囲まれた文字列で、直後に B が付きます。0 は、ビットがオフであるかどうかをテストします。1 は、ビットがオンであるかどうかをテストし、ピリオドはビットを無視します。ビットマスクにカンマ(,)とブランクを挿入すると、意味を変えずに読みやすくすることができます。

注意:
: **ビットマスクを使用すると、切り捨てが実行されることがあります。** ビットマスクより長い式は、ビットマスクとの比較が行われる前に、自動的に切り捨てられます。このため、
  ```

比較の結果が偽になる可能性があります。式の長さ(ビット数)がビットマスクの長さを超えないようにする必要があります。ビットマスクが文字式より長い場合は、SAS ログに警告メッセージが表示されます。ビットマスクの左側が切り捨てられ、処理は継続されます。

次の例では、文字変数をテストしています。

```
if a='..1.0000'b then do;
```

A の左から 3 番目のビットがオンで、5 - 8 番目のビットがオフの場合、比較の結果は真になり、式の結果は 1 になります。これ以外の場合、比較の結果は偽になり、式の結果は 0 になります。より詳細な例を次に示します。

```
data test;
  input @88 bits $char1.;
  if bits='10000000'b
    then category='a';
  else if bits='01000000'b
    then category='b';
  else if bits='00100000'b
    then category='c';

run;
```

注: 割り当てステートメントでは、ビットマスクをビットテスト定数として使用することはできません。たとえば、次のステートメントは無効です。

```
x='0101'b;    /* incorrect*/
```

ビットテストを行う場合、$BINARY*w* および BINARY*w* 出力形式、ならびに $BINARY*w*、BINARY*w.d*、および BITS*w.d* 入力形式が役立ちます。これらの入出力形式を使用することで、文字値や数値をバイナリ値に変換したり、バイナリ値を文字値や数値に変換したりできます。また、入力データから特定のビットを抽出することもできます。出力形式と入力形式の詳細については、*SAS Formats and Informats: Reference* を参照してください。

定数のよくあるエラーを避ける

変数名の前に引用符で囲んだ文字列を指定する場合、閉じ引用符と変数名の間にブランクを必ず挿入してください。ブランクを挿入していない場合、SAS は変数名の前の文字定数を特殊な SAS 定数として変換する可能性があります(次の表を参照)。

表 6.2 *文字定数の後にある場合に解釈の誤りの原因となる文字*

文字定数の後の文字	起こりうる解釈の誤り	例
b	ビットテスト定数	'00100000'b
d	日付定数	'01jan04'd
dt	日時定数	'18jan2005:9:27:05am'dt
n	名前リテラル	'My Table'n
t	時間定数	'9:25:19pm't
x	16 進表記	'534153'x

次の例では、'821't は時間定数として評価されます。SAS 時間定数の詳細については、“日付定数、時間定数、日時定数” (89 ページ)を参照してください。

```
data work.europe;
   set ia.europe;
   if flight='821'then
      flight='230';
run;
```

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。

ログ 6.1 *時間リテラルの解釈の誤りによって引き起こされるエラーのログ結果*

```
ERROR:Invalid date/time/datetime constant '821't.ERROR 77-185:Invalid number
conversion on '821't.ERROR 388-185:Expecting an arithmetic operator.
```

IF ステートメントの閉じ引用符とその次の文字の間にブランクを挿入すると、この解釈の誤りが修正されます。エラーメッセージは生成されず、FLIGHT 値 821 を持つすべてのオブザベーションが 230 に置き換えられます。

```
if flight='821' then
   flight='230';
```

式内の SAS 変数

定義

変数
: 変数は、所定の特性を示すデータ値の集合です。SAS 式中で変数を使用できます。

数値と文字の自動変換

SAS 式内で変数を指定する場合、その変数値の種類が呼び出し対象の変数値の種類と一致しないときは、値は適切な種類へと自動的に変換されます。次の規則に従って、文字変数は数値変数に、数値変数は文字変数に自動的に変換されます。

- 正符号(+)など、数値オペランドを必要とする演算子とともに文字変数を使用した場合、文字変数は数値に変換されます。
- 等号(=)などの比較演算子を使用して文字変数と数値変数を比較した場合、文字変数は数値に変換されます。
- 連結演算子など、文字値を必要とする演算子とともに数値変数を使用した場合、数値は BEST12.出力形式を使用して文字に変換されます。変換結果は右端のバイトから順番に格納されるので、BEST12.出力形式を十分に格納できる長さを持つ変数を用いて、変換された値を格納する必要があります。変換結果は、LEFT 関数を使用して左揃えにすることができます。
- 割り当てステートメントの左辺で数値変数を使用し、右辺で文字変数を使用した場合、文字変数は数値に変換されます。逆に、文字変数が左辺にあって数値変数が右辺にある場合は、数値変数が BESTn.出力形式(n は左辺の変数の長さ)を使用して文字に変換されます。

自動変換が実行されると、変換が行われたことを通知するメッセージが SAS ログに表示されます。文字変数を数値に変換した結果が無効な数値になる場合は、変換結果として欠損値が割り当てられ、SAS ログにエラーメッセージが出力されるとともに、自動変数_ERROR_の値に 1 が設定されます。

注: データ値の変換には PUT 関数と INPUT 関数を使用することもできます。これらの関数を使用すると、自動変換よりも効率が良くなる場合があります。PUT 関数の例については、“連結演算子” (101 ページ) を参照してください。これらの関数の詳細については、*SAS Functions and CALL Routines: Reference* を参照してください。

SAS 変数の詳細については、4 章, “SAS 変数” (35 ページ)を参照してください。

式内の SAS 関数

SAS 関数は、特定の演算またはシステム操作を実行するときに使用するキーワードです。SAS 関数は SAS 式の中で使用でき、値を返します。また、引数が必要な関数もあります。SAS 機能の詳細については、*SAS Functions and CALL Routines: Reference* を参照してください。

式内の SAS 演算子

定義

SAS 演算子は、比較演算、算術演算、論理演算を行うための記号です。SAS 関数や、グループ化のためのかっこも演算子の一種です。SAS System で使用される演算子には、大きく分けると次の 2 種類があります。

- 接頭演算子
- 挿入演算子

接頭演算子は、変数、定数、関数、またはかっこ付きの式の直前に記述する演算子です。接頭演算子として使用できるのは、正符号(＋)と負符号(－)です。NOT というワード(ニーモニック)と、それに対応する記号も接頭演算子になります。次の例では、接頭演算子を変数、定数、関数、かっこ付きの式とともに使用しています。

- `+y`
- `-25`
- `-cos(angle1)`
- `+(x*y)`

挿入演算子は、6<8 の<などのように、オペランドの間に記述する演算子です。挿入演算子には、次の種類があります。

- 算術演算子
- 比較演算子
- 論理(ブール)演算子
- 最小演算子
- 最大演算子

- 連結演算子

算術演算で使用する場合は、正符号(＋)と負符号(－)も挿入演算子になります。

また、特定の SAS ステートメントでのみ使用される演算子もあります。WHERE ステートメントには、WHERE 式とともに使用する場合だけ有効になる、特殊な SAS 演算子があります。それらの演算子の詳細については、11 章, “WHERE 式の処理” (165 ページ)を参照してください。_NEW_演算子を使用すると、DATA ステップコンポーネントオブジェクトのインスタンスを作成できます。詳細については、22 章, “DATA ステップコンポーネントオブジェクトの使用” (507 ページ)を参照してください。

算術演算子

算術演算子は、次の表に示すとおり、算術計算の実行を指示します。

表 6.3 算術演算子

記号	定義	例	結果
**	累乗	`a**3`	A を 3 乗します。
*	乗算*	`2*y`	2 に Y の値を掛けます。
/	除算	`var/5`	VAR の値を 5 で割ります。
+	加算	`num+3`	NUM の値に 3 を足します。
-	減算	`sale-discount`	SALE の値から DISCOUNT の値を引きます。

* 乗算を指示するときは、アスタリスク(*)が常に必要です。2Y や 2(Y)は有効な式ではありません。

算術演算子のオペランドが欠損値である場合は、演算結果も欠損値になります。欠損値のプロパゲーションを防止する方法については、5 章, “欠損値” (77 ページ) を参照してください。

SAS がこれらの演算子を評価する順番については、“複合式の評価順序” (102 ページ) を参照してください。

比較演算子

比較演算子は、2 つの変数、定数、または式を使用して、比較、操作、計算を指示する演算子です。比較演算が真の場合、結果は 1 になります。比較演算が偽の場合には、結果は 0 になります。

比較演算子は、次の表に示すとおり、記号、または対応するニーモニックで表すことができます。

表 6.4 比較演算子

記号	ニーモニック	定義	例
=	EQ	等しい	`a=3`
^=	NE	等しくない*	`a ne 3`
¬=	NE	等しくない	
~=F	NE	等しくない	
>	GT	より大きい	`num>5`
<	LT	より小さい	`num<8`
>=	GE	以上**	`sales>=300`
<=	LE	以下***	`sales<=100`
	IN	リストのどれかと等しい	`num in (3, 4, 5)`

* NE に使用する記号は、PC によって異なります。

** SAS System の以前のバージョンとの互換性を保つために、記号=>も指定できます。これは WHERE 句や PROC SQL ではサポートされません。

*** SAS System の以前のバージョンとの互換性を保つために、記号=<も指定できます。これは WHERE 句や PROC SQL ではサポートされません。

SAS がこれらの演算子を評価する順番については、“複合式の評価順序” (102 ページ) を参照してください。

演算子の後ろにコロン修飾子(:)を追加すると、文字列の接頭部だけを比較できます。詳細については、“文字の比較” (97 ページ) を参照してください。

IN 演算子

IN 演算子を使用すると、演算子の左側の式で生成される値を、右側の値のリストと比較できます。個々の値は、ブランクまたはカンマで区切ることができます。コロンを使用すると、連続した整数の範囲を指定できます。

IN 比較の形式には次の 3 つがあります。

expression IN(*value-1*<...,*value-n*>)
expression IN(*value-1*<... *value-n*>)
expression IN(*value-1*<...:*value-n*>)

比較するコンポーネントは次のとおりです。

式
有効な SAS 式を使用できますが、通常は IN 演算子で使用されるときには変数名です。

値
定数の必要があります。

IN 演算子の詳細と例については、“数値比較における IN 演算子” (96 ページ)を参照してください。

数値の比較

数値の比較演算は、値を基準にして行われます。A<=B という式で、A の値が 4、B の値が 3 の場合、A<=B の値は 0 で偽になります。A の値が 5、B の値が 9 の場合、この式の値は 1 で真になります。A と B の値がいずれも 47 である場合には、式は真になり、値は 1 になります。

比較演算子は、次の例のように、IFTHEN ステートメントの中でよく使用されます。

```
if x<y then c=5;
   else c=12;
```

比較演算は、割り当てステートメントの式の中でも実行できます。たとえば、前述のステートメントは、次のように書き換えることができます。

```
c=5*(x<y)+12*(x>=y);
```

かっこ内の数値は他の演算より前に評価されるので、式 **(x<y)** と **(x>=y)** が最初に評価されます。かっこ内の式は、演算の結果(1 または 0)で置き換えられます。したがって、X=6 で Y=8 の場合、式は次のように評価されます。

```
c=5*(1)+12*(0)
```

このステートメントの処理結果は C=5 になります。

長さの異なる数値を比較すると、正しい結果が得られない場合があります。これは、8 バイト未満の値の精度は 8 バイトより長い値の精度よりも低いためです。値の丸めも、数値の比較演算の結果に影響を与えます。数値精度の詳細な説明については、4 章, “SAS 変数” (35 ページ) を参照してください。

数値欠損値は他の数値よりも小さく、独自の並べ替え順序を持っています。詳細については、5 章, “欠損値” (77 ページ) を参照してください。

数値比較における IN 演算子

簡略表記を使用して、検索する連続した整数の範囲を指定できます。この範囲を指定するには、検索対象とするリスト内の値として構文 M:N を使用します。ここで、M は下限、N は上限になります。M および N は整数でなければなりません。M、N、および M と N の間にあるすべての整数が範囲に含まれます。たとえば、次の 2 つのステートメントは同じ意味を持ちます。

- **y = x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);**
- **y = x in (1 2 3 4 5 6 7 8 9 10);**
- **y = x in (1:10);**

同一の IN リスト内で複数の範囲を使用できます。また、1 つの IN リスト内で範囲と通常の定数を組み合わせて使用することもできます。次の例では、1 つの IN リスト内で範囲指定と通常の定数を組み合わせて使用することにより、X が 0、1、2、3、4、5、9 であるかどうかを判定しています。

```
if x in (0,9,1:5);
```

IN 演算子を使用すると、数値の配列を検索できます。たとえば、次のコードでは、配列 **a** が作成され、定数 **x** を定義し、IN 演算子を使用して **x** を配列 **a** で検索します。**array a{10} (2*1:5)** の配列初期化構文では、1、2、3、4、5、1、2、3、4、5 の初期値を含む配列を作成します。

```
   data _null_;
```

```
    array a{10} (2*1:5);
    x=99;
    y = x in a;
    put y=;
    a{5} = 99;
    y = x in a;
    put y=;
run;
```

ログ 6.2 *IN 演算子を使用して数値の配列を検索した結果(一部)*

```
173 data _null_; 174 array a{10} (2*1:5); 175 x=99; 176 y = x in a; 177 put y=;
178 a{5} = 99; 179 y = x in a; 180 put y=; 181 run; y=0 y=1
```

注: PROC SQL はこの構文をサポートしていません。

文字の比較

比較演算は文字オペランドに対しても実行できますが、比較の結果は常に数値(1 または 0)になります。文字オペランドは、左から右へ 1 文字ずつ比較されます。文字の順序は、コンピュータで使用されている照合順序(通常は ASCII または EBCDIC)によって決定します。

たとえば、ASCII 照合順序と EBCDIC 照合順序では、**G** は **A** よりも大きいものと見なされます。したがって、次の式は真になります。

```
'Gray'>'Adams'
```

長さの異なる 2 つの文字値を比較する場合は、短い方の値の末尾に、不足している分だけブランクが補われます。比較演算はその後に開始されます。ブランクの文字欠損値は、他の表示可能な文字値よりも小さいものと評価されます。たとえば、**.** は **h** よりも小さいため、次の式は真になります。

```
'C. Jones'<'Charles Jones'
```

後置ブランクは比較演算では無視されるため、**'fox '** (後置ブランクあり)と**'fox'**は等価となります。ただし、文字値の先頭や途中にあるブランクは SAS にとって意味があるため、**' fox'** (前置ブランクあり)と**'fox'**は等価ではありません。

比較演算子の後ろにコロン(:)を使用すると、文字式の一定の先頭部分だけを比較できます。比較演算中に、長い方の値は短い方の値に合わせて切り捨てられます。次の例では、等号(=)の後ろにコロン修飾子(:)を記述することによって、変数 LastName の値のうち、先頭の文字だけを評価しています。ここでは、名前が **S** で始まるオブザベーションが選択されます。

```
if lastname=:'S';
```

表示可能な文字はブランクより大きいので、次のどちらのステートメントを使用しても、変数 LastName の値が文字 **S** 以上であるオブザベーションが選択されます。

- **if lastname>='S';**
- **if lastname>=:'S';**

注: 長さがゼロの文字値を、IN:比較演算または EQ:比較演算で他の文字値と比較した場合、2 つの文字値は等価とは見なされません。結果は、常に 0 (偽)になります。

ここでは、文字列の全体、および文字列の先頭を比較する方法について説明しました。SAS 文字関数を使用すると、文字列の一部の値を検索対象にしたり、抽出したり

できます。すべての SAS 関数の詳細については、*SAS Functions and CALL Routines: Reference* を参照してください。

文字の比較における IN 演算子

IN 演算子を文字列とともに使用すると、指定した文字値のリストの中に変数の値が含まれているかどうかを判定できます。次の 2 つのステートメントでは、同じ結果が得られます。

- **if state in ('NY','NJ','PA') then region+1;**
- **if state in ('NY' 'NJ' 'PA') then region+1;**
- **if state='NY' or state='NJ' or state='PA' then region+1;**

IN 演算子を使用すると、文字値の配列を検索できます。たとえば、次のコードでは、配列 **a** が作成され、定数 **x** を定義し、IN 演算子を使用して **x** を配列 **a** で検索します。

```
data _null_;
   array a{5} $ (5*'');
   x='b1';
   y = x in a;
   put y=;
   a{5} = 'b1';
   y = x in a;
   put y=;
run;
```

ログ 6.3 *IN 演算子を使用して文字値の配列を検索した結果(一部)*

```
190 data _null_; 191 array a{5} $ (5*''); 192 x='b1'; 193 y = x in a; 194 put
y=; 195 a{5} = 'b1'; 196 y = x in a; 197 put y=; 198 run; y=0 y=1
```

論理(ブール)演算子と式

論理演算子は、ブール演算子とも呼ばれ、通常、複数の比較演算式を連結する場合に使用されます。論理演算子を次に示します。

表 6.5 *論理演算子*

記号	ニーモニック	例
&	AND	**(a>b & c>d)**
|	OR*	**(a>b or c>d)**
!	OR	
¦	OR	
¬	NOT**	**not(a>b)**
^	NOT	

記号	ニーモニック	例
~	NOT	

* OR に使用する記号は、動作環境によって異なります。
** NOT に使用する記号は、動作環境によって異なります。

これらの演算子を SAS が評価する順序については、“複合式の評価順序” (102 ページ)を参照してください。

また、論理演算子を含まない数式は、ブール式として使用できます。ブール式の例は、“ブール式” (100 ページ)を参照してください。

AND 演算子

AND で連結された数値がいずれも 1 (真)の場合、AND 演算の結果は 1 (真)になります。そうでない場合、結果は 0 (偽)になります。次に比較演算の例を示します。

```
a<b& c>0
```

この例では、A<B と C>0 がいずれも 1 (真)のときに限り、結果が真(値が 1)になります。つまり、A が B より小さく、かつ、C が正の数値である場合です。

共通の変数を持つ 2 つの比較演算式を AND で連結する場合は、暗黙的な AND を使用して、式を短縮することができます。たとえば、次の 2 つのサブセット化 IF ステートメントでは、同じ結果が得られます。

- **if 16<=age and age<=65;**
- **if 16<=age<=65;**

OR 演算子

OR で連結された数値のいずれかが 1 (真)の場合、OR 演算の結果は 1 (真)になります。そうでない場合、結果は 0 (偽)になります。次に比較演算の例を示します。

```
a<b|c>0
```

A<B が 1 (真)の場合は、C の値にかかわらず、結果は真(値は 1)になります。C>0 の値が 1 (真)の場合は、A や B の値にかかわらず、結果は真になります。したがって、その関係の片方または両方が維持されている場合は、真になります。

IF ステートメント、SELECT ステートメント、WHERE ステートメントなどで、OR 演算子を一連の比較演算とともに使用するときは、注意が必要です。一連の OR 比較演算が 1 つでも真になれば、条件は真になりますし、0 以外の、非欠損値である定数は常に真として評価されます。SAS によるブール式の計算方法の詳細については、“ブール式” (100 ページ)を参照してください。たとえば、次のサブセット化 IF ステートメントの比較演算は、常に真になります。

```
if x=1 or 2;
```

最初に X=1 が評価されます。その結果は真または偽のいずれかです。しかし、2 が 0 以外の非欠損値(真)として評価されるので、式全体は真になります。次のステートメントでは、どちらの比較演算も真または偽として評価され得るので、条件は必ずしも真にはなりません。

```
if x=1 or x=2;
```

NOT 演算子

接頭演算子 NOT は、論理演算子でもあります。値が 0 (偽)である数値の先頭に NOT を指定すると、結果は 1 (真)になります。偽のステートメントを否定するので、結果は 1 (真)です。たとえば、X=Y が 0 (偽)の場合、NOT(X=Y)は 1 (真)になります。値が欠損している数値の先頭に NOT を指定しても、結果は 1 (真)になります。0 以外の非欠損値を含んだ数値の先頭に NOT を指定すると、結果は 0 (偽)になります。真のステートメントを否定するので、結果は 0 (偽)です。

たとえば、次の 2 つの式は同じ意味を持ちます。

- **not(name='SMITH')**
- **name ne 'SMITH'**

また、NOT(A&B)は NOT A | NOT B と等しく、NOT(A|B)と NOT A & NOT B は同じです。たとえば、次の 2 つの式は同じ意味を持ちます。

- **not(a=b & c>d)**
- **a ne b | c le d**

ブール式

コンピュータによる論理演算では、真の値は 1、偽の値は 0 です。0 以外の数値や非欠損値は真で、0 や欠損値は偽になります。したがって、数値変数や数式を単独で条件式として使用できます。それらの値が 0 以外の数値または非欠損値の場合、条件式は真になります。その値が 0 または欠損値の場合、条件は偽になります。

```
0 | . = False
1 = True
```

たとえば、特定のオブザベーションに Cost の値が存在するかどうかによって、変数 Remarks に値を入力するとします。この場合は、次のような IFTHEN ステートメントを記述できます。

```
if cost then remarks='Ready to budget';
```

このステートメントは次に相当します。

```
if cost ne . and cost ne 0
   then remarks='Ready to budget';
```

数式のかわりに、次のように単純な数値定数を置くこともできます。

```
if 5 then do;
```

関数によって返される数値は、数式としても有効です。

```
if index(address,'Avenue') then do;
```

MIN 演算子と MAX 演算子

最小(MIN)演算子と最大(MAX)演算子は、2 つの数値を比較して最小値または最大値を見つけるために使用します。これらの演算子は、最小値または最大値を調べる 2 つの数値で囲んで指定します。MIN (><)演算子は、2 つの値を比較して小さい方の値を返します。MAX (<>)演算子は、2 つの値を比較して大きい方の値を返します。たとえば、A<B である場合、A><B は値 A を返します。

比較演算に欠損値が含まれている場合は、“欠損値の順序” (79 ページ)で説明されている欠損値の並べ替え順序が使用されます。たとえば、.A<>.Z という演算を行ったときに返される最大値は、値.Z になります。

注: WHERE ステートメントまたは句では、<>演算子は NE と同じです。

連結演算子

連結演算子(||)は、文字値を連結します。連結演算の結果は、**level='grade'||'A'**.などのように、割り当てステートメントを使用する変数に格納するのが一般的です。結果として得た変数の長さは、連結演算に関係している変数または定数の長さの合計になります。ただし、LENGTH ステートメントまたは ATTRIB ステートメントを使用して、新しい変数の長さを指定することもできます。

連結演算子は、前置ブランクまたは後置ブランクを切り捨てません。変数が後置ブランクで埋められている場合は、連結する前に変数の長さをチェックし、TRIM 関数を使用して値の後置ブランクを削除してください。追加文字関数の詳細と例については、*SAS Functions and CALL Routines: Reference* を参照してください。

たとえば、次の DATA ステップでは、変数 Color の長さが 8 であるため、連結後の値にはブランクが含まれています。

```
data namegame;
   length color name $8 game $12;
   color='black';
   name='jack';
   game=color||name;
   put game=;
run;
```

Game の値は**'black jack'**です。挿入されているブランクを削除するには、次のように、連結演算で TRIM 関数を使用します。

```
game=trim(color)||name;
```

このステートメントでは、変数 Game の値は**'blackjack'**になります。連結演算子の使用方法は他にもあります。次に例を示します。

- A の値が**'fortune'**、B の値が**'five'**、C の値が**'hundred'**である場合、次のステートメントでは、変数 D の値は**'fortunefivehundred'**になります。

  ```
  d=a||b||c;
  ```

- この例では、変数の値と文字定数を連結しています。

  ```
  newname='Mr.or Ms. ' ||oldname;
  ```

  OldName の値が**'Jones'**である場合、NewName の値は**'Mr. or Ms. Jones'**になります。

- 連結演算ではブランクが削除されないため、次の式では値は、**'JOHN SMITH'** (ブランクの埋め込みあり)になります。

  ```
  name='JOHN   '||'SMITH';
  ```

- この例では、PUT 関数を使用して、数値を文字値に変換しています。さらに、TRIM 関数を使用して、ブランクを削除しています。

  ```
  month='sep   ';
  year=99;
  date=trim(month) || left(put(year,8.));
  ```

  変数 DATE の値は、**'sep99'**という文字値になります。

複合式の評価順序

表 6.6 (102 ページ) に、複合式の評価順序を示します。表の各項目が示す内容は次のとおりです。

優先順位
評価の優先順位に従って、SAS 演算子を 7 つにグループ化したものです。複合式では、グループ I の演算子を含んでいる部分が最初に評価され、以降は、優先順位に従って各グループが評価されます。

評価の順序
式の中で最初に評価される部分を決める規則です。複合式では、かっこを使用してオペランドをまとめることがあります。かっこ内の式は、かっこの外側にある式よりも先に評価されます。これらの規則は、複合式の中で同じグループの演算子を複数使用する場合の評価順序にも適用されます。

記号
比較演算、算術演算、論理演算を表す演算子の記号です。

ニーモニック
演算子のニーモニックです。ニーモニックは、使用しているキーボードがサポートしていない特殊記号がある場合などに使用します。

定義
記号の定義です。

例
SAS 式で記号またはニーモニックを使用する方法を、簡単な例で示しています。

表 6.6 複合式の評価順序

優先順位	評価の順序	記号	ニーモニック	定義	例
グループ I	右から左	**		累乗*	`y=a**2;`
		+		正符号**	`y=+(a*b);`
		-は指定できない		負符号***	`z=-(a+b);`
		◦ ¬ ~	NOT	論理否定†	`if not z then put x;`
		><	MIN	最小演算子††	`x=(a><b);`
		<>	MAX	最大演算子	`x=(a<>b);`
グループ II	左から右	*		乗算	`c=a*b;`
		/		除算	`f=g/h;`
グループ III	左から右	+		加算	`c=a+b;`
		-は指定できない		減算	`f=g-h;`
グループ IV	左から右	|| ¦¦ !!		文字値の連結†††	`name= 'J'||'SMITH';`

優先順位	評価の順序	記号	ニーモニック	定義	例
グループ V‡	左から右‡‡	<	LT	より小さい	`if x<y then c=5;`
		<=	LE	以下	`if x le y then a=0;`
		=	EQ	等しい	`if y eq (x+a) then output;`
		¬=	NE	等しくない	`if x ne z then output;`
		>=	GE	以上	`if y>=a then output;`
		>	GT	より大きい	`if z>a then output;`
			IN	リストのどれかと等しい	`if state in ('NY','NJ','PA') then region='NE';` `y = x in (1:10);`
グループ VI	左から右	&	AND	論理積	`if a=b & c=d then x=1;`
グループ VII	左から右	| ¦ !	OR	論理和‡‡‡	`if y=2 or x=3 then a=d;`

* グループ I の演算子は右から左へ評価されるので、x=2**3**4 という式は x=(2**(3**4))として評価されます。

** 正符号(+)は、接頭演算子としても算術演算子としても使用されます。正符号(+)が接頭演算子になるのは、式の先頭にある場合と、開きかっこや別の演算子の直後にある場合だけです。

*** 負符号(-)は、接頭演算子としても算術演算子としても使用されます。負符号(-)が接頭演算子になるのは、式の先頭にある場合と、開きかっこや別の演算子の直後にある場合だけです。

† キーボードで入力できる文字に応じて、ノット記号(¬)、チルダ(~)、キャレット(^)を論理否定の記号として使用できます。SAS システムオプションの CHARCODE を使用すると、使用できない特殊文字の代用となるさまざまな記号を使用できます。

†† たとえば、-3><-3 は-(3><-3)として評価されます。この式は-(-3)、つまり+3 と等しくなります。このような結果になるのは、グループ I の演算子は右から左へと評価されるためです。

††† キーボードで入力できる文字に応じて、2 つの実線の縦棒(||)、切れ目のある縦棒(¦¦)、感嘆符(!!)を連結演算子として使用できます。

‡ グループ V の演算子は、比較演算子です。比較演算の結果は、比較が真であれば 1、偽であれば 0 です。欠損値は、どのような比較演算でも最小値として扱われます。SAS System の以前のバージョンとの互換性を保つために、=< (以下)という記号も使用できます。文字を比較するときは、比較演算子の後ろにコロン(:)を付加することで、値の先頭にある 1 文字だけ、または何個かの文字だけを比較できます。比較演算中に、長い方の値は短い方の値に合わせて切り捨てられます。たとえば、name=:'P'と記述すると、NAME の先頭にある文字の値が文字の P と比較されます。

‡‡ 2 つの比較演算子が数値を囲んでいる場合は例外です。たとえば、x<y<z という式は(x<y)と(y<z)として評価されます。

‡‡‡ キーボードで入力できる文字に応じて、実線の縦棒(|)、切れ目のある縦棒(¦)、感嘆符(!)を論理和の記号として使用できます。また、OR の記号に対応するニーモニックも使用できます。

7 章

日付、時間と間隔

SAS 日付値、時間値、日時値について

定義

SAS 日付値
SAS 日付値は、1960 年 1 月 1 日から指定した日付までの経過日数を表す値です。SAS System では、西暦 1582 年 11 月から西暦 19,900 年までの範囲で日付計算を実行できます。1960 年 1 月 1 日より以前の日付は負の数、それ以降の日付は正の数で表します。

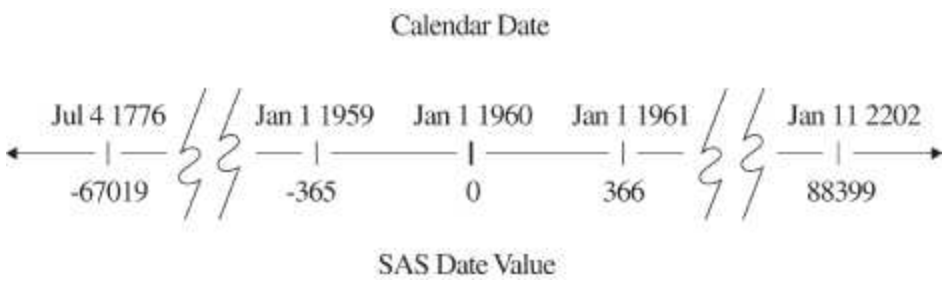
- SAS 日付値は、2000 年のうるう日を含むすべてのうるう日に対応しています。
- SAS 日付値を利用すると、特定の日が何曜日にあたるかを、1752 年 9 月まで遡って調べることができます。この時期に、数日間を削除することによって暦が調整されました。将来における SAS の曜日と時間の長さは、西暦 19,900 年まで正確に計算できます。
- さまざまな SAS 言語要素が SAS 日付値(関数、出力形式、入力形式)を処理します。

SAS 時間値
当日午前 0 時からの経過秒数を表す値です。SAS 時間値の範囲は 0 - 86400 です。

SAS 日時値
1960 年 1 月 1 日から指定した日付までの、経過秒数を表す値です。日付と一緒に時、分、秒を指定することもできます。

次の図は、カレンダ形式の日付と SAS 日付値の対応を表したものです。

図 7.1 *カレンダ日付から SAS 日付値への変換方法*



2 桁と 4 桁の年

SAS System では、2 桁または 4 桁の年の値を読み込むことができます。データに 2 桁の年が含まれている場合は、YEARCUTOFF=システムオプションを使用することで、その年がどの世紀に該当するのかを指定できます。たとえば、YEARCUTOFF=1950 は 2 桁の年 50 - 99 が 1950 - 1999 年に対応することを示します。2 桁の年 00 - 49 は 2000 - 2049 年に対応します。SAS System では、YEARCUTOFF=オプションのデフォルト値が 1926 に設定されていますが、DATA ステップの YEARCUTOFF=値を調整することで、そのとき処理している日付値の範囲に合わせることができます。2000 - 2099 年までの日付を表す 2 桁の年を正しく処理するには、YEARCUTOFF=システムオプションに 1901 - 2000 までの適切な値を指定する必要があります。詳細については、“YEARCUTOFF= System Option” (*SAS System Options: Reference*)を参照してください。

5 桁の年

一部の形式では DATETIME20.などの 5 桁の年を十分に格納できる幅を指定しますが、SAS ドキュメントには 5 桁の年が表示されません。

2000 年の扱い

YEARCUTOFF=システムオプションの使用

SAS System では、2000 年を他のうるう年と同様に扱います。年を表す 2 桁の数字を日付に使用する場合は、操作するデータの日付範囲に合わせて、YEARCUTOFF=システムオプションのデフォルト設定を調整する必要があります。調整しない場合は、4 桁の年を使用してください。次のプログラムでは、YEARCUTOFF=システムオプションの値を 1950 に変更しています。この変更によって、2 桁の日付はすべて 1950 - 2049 までの 100 年間に属するものと見なされます。

```
options yearcutoff=1950;
data _null_;
   a='26oct02'd;
```

```
   put 'SAS date='a;
   put 'formatted date='a date9.;
run;
```

PUT ステートメントは、次の行を SAS ログに書き込みます。

```
SAS date=15639
formated date=26OCT2002
```

注: できるだけ、年は 4 桁で指定してください。4 桁の年の値は、ほとんどの SAS 日時言語要素でサポートされています。

例:YEARCUTOFF=が 2 桁と 4 桁の年に与える影響

次の例では、データ中に 2 桁と 4 桁の年が混在している場合の処理結果を示します。デフォルトでは、YEARCUTOFF=オプションは 1926 に設定されています。

```
   options nodate;


   data schedule;
      input @1 jobid $ @6 projdate mmddyy10.;
      datalines;
   A100 01/15/25
   A110 03/15/2025
   A200 01/30/96
   B100 02/05/12
   B200 06/15/2012
   ;

   proc print data=schedule;
      format projdate mmddyy10.;
   run;
```

PRINT プロシジャの出力結果は次のようになります。

アウトプット 7.1 *YEARCUTOFF=を 1926 に設定した結果の 4 桁の年を示す出力*

The SAS System

Obs	jobid	projdate
1	A100	01/15/2025
2	A110	03/15/2025
3	A200	01/30/1996
4	B100	02/05/2012
5	B200	06/15/2012

この例では、注目すべき点がいくつかあります。

- この DATA ステップのデータ行では、1 行目に 2 桁の年 25 が含まれ、2 行目に 4 桁の年 2025 が含まれています。2025 が 1926–2025 の範囲内なので、1 行目の世紀はデフォルトで 2000 年代になります。2 行目の 4 桁の年は、YEARCUTOFF=オプションの影響を受けません。

- 3 行目では、1996 年が 1926–2025 の範囲内なので、世紀はデフォルトで 1900 年代になります。
- 4 行目と 5 行目の出力は、1 行目と 2 行目と同じような結果になります。4 行目には 2 桁の年 12 が含まれ、5 行目には 4 桁の年 2012 が含まれています。2012 が 1926–2025 の範囲内なので、4 行目の世紀はデフォルトで 2000 年代になります。5 行目の 4 桁の年は、YEARCUTOFF=オプションの影響を受けません。

前述のように、2 桁の年を指定した場合は、年の上位 1 - 2 桁が、意図した結果と異なる場合があります。YEARCUTOFF=システムオプションの最適値は、処理の対象となる日付の範囲によって異なります。

YEARCUTOFF=システムオプションのデフォルト値は、SAS System のバージョン 6.06 - 6.12 では 1900 です。SAS 7 からはデフォルト値が 1920、バージョン 9.4 からはデフォルト値が 1926 に変更されました。

SAS での日付処理のしくみの詳細については、日付値、時間値、日時値に関するセクションを参照してください。

日付に関する一貫性を確保する方法

データ処理で日付を扱う場合に、日付値が常に正しく変換されるようにするには、次の点に注意します。

- 日付は、単純な数値や文字値としてではなく、SAS 日付値として格納します。
- 2 桁の年が含まれる日付を SAS 日付値に変換するときは、YEARCUTOFF=システムオプションを使用します。
- 読み込むデータセットを検査して、2 桁の年を含む日付が YEARCUTOFF=システムオプションに基づいてすべて正しく解釈されるかどうかを確認します。次の状況を確認します。
  - 2 桁の年が 100 年間の範囲内に収まっているかどうかを確認します。日付が 100 年間の範囲内に収まっていない場合は、データに 4 桁の年を使用するか、DATA ステップ内で条件付きロジックを使用して、日付が正しく解釈されるようにする必要があります。
  - YEARCUTOFF=システムオプションのデフォルトの範囲を調整しなければならないような 2 桁の年があるかどうかを確認します。たとえば、動作環境で YEARCUTOFF=システムオプションのデフォルト値が 1926 に設定され、データに 1925 年を表す 2 桁の年が含まれている場合、この値の処理に使用する SAS プログラムでは、YEARCUTOFF=システムオプションの値を 1 年分小さくして調整する必要があります。
- 出力 SAS データセットの中では、日付が SAS 日付値として表されていることを確認します。
- SAS プログラムをチェックして、2 桁の年を使用する出力形式や入力形式 (DATE7.、MMDDYY6.、MMDDYY8.など)を使用したときに、データの読み込みや書き出しが正しく行われるかどうかを確認します。

注: YEARCUTOFF=システムオプションは、すでに SAS 日付値として格納されている日付には作用しません。

SAS 日付値と SAS 時間値の操作

入力形式と出力形式

SAS System では、出力形式および入力形式と呼ばれる SAS 言語要素によって、日付値、時間値、日時値を、カレンダ形式の日付や時計時間に相互に変換できます。

- 変換には、SAS 入力形式と SAS 出力形式を使用します。SAS 出力形式は、SAS System で認識される日付値や時間値などの値を、長さや表記がさまざまに異なるカレンダ形式の日付や時計時間として出力します。
- SAS 入力形式は、時計時間やカレンダ形式の日付など、表記や長さがさまざまに異なる値を読み込んで、SAS 日付値、SAS 時間値、SAS 日時値に変換します。

SAS は、次の文字で区切られた日付値と時間値を読み込むことができます。

!# $ % & ( ) * + - ./ :; < = > ?[\ ] ^ _ { | } ~

ブランク文字も使用できます。

日付には、1 つの区切り文字のみを使用できます。複数の区切り文字を使用すると、エラーメッセージが SAS ログに書き出されます。たとえば、01/Jan/2007 では区切り文字が 1 つなので、SAS で読み込み可能です。01-Jan/2007 の場合、日付の区切り文字が 2 つなので、エラーメッセージが書き出されます。

タスク別の日時ツール

SAS では、日付や時間のデータをさまざまな形式で操作することができます。次の表に、日時の操作をタスク別に示します。

表 7.1 日時に関するタスク(その 1)

タスク	言語要素のタイプ	言語要素	入力	結果
SAS 日付値の書き出し	日付の出力形式	DATE.	19434	17MAR13
		DATE9.	19434	17MAR2013
		DAY.	19434	17
		DDMMYY.	19434	17/03/13
		DDMMYY10.	19434	17/03/2013
		DDMMYYB.	19434	17 03 13
		DDMMYYB10.	19434	17 03 2013
		DDMMYYC.	19434	17:03:13
		DDMMYYC10.	19434	17:03:2013
		DDMMYYD.	19434	17-03-13
		DDMMYYD10.	19434	17-03-2013
		DDMMYYN.	19434	17032013
		DDMMYYN6.	19434	170313
		DDMMYYP.	19434	17.03.13
		DDMMYYP10.	19434	17.03.2013
		DDMMYYS.	19434	17/03/13
		DDMMYYS10.	19434	17/03/2013
		DOWNAME.	19434	Sunday
		JULDAY. *	19434	76
		JULIAN. *	19434	13076
		MMDDYY.	19434	03/17/13
		MMDDYY10.	19434	03/17/2013
		MMDDYYB.	19434	03 17 13
		MMDDYYB10.	19434	03 17 2013
		MMDDYYC.	19434	03:17:13
		MMDDYYC10.	19434	03:17:2013
		MMDDYYD.	19434	03-17-13

タスク	言語要素のタイプ	言語要素	入力	結果
SAS 日付値の書き出し	日付の出力形式	MMDDYYD10.	19434	03-17-2013
		MMDDYYN.	19434	03172013
		MMDDYYN8.	19434	03172013
		MMDDYYP.	19434	03.17.13
		MMDDYYP10.	19434	03.17.2013
		MMDDYYS.	19434	03/17/13
		MMDDYYS10.	19434	03/17/2013
		MMYY.	19434	03M2013
		MMYYC.	19434	03:2013
		MMYYD.	19434	03-2013
		MMYYN.	19434	032013
		MMYYP.	19434	03.2013
		MMYYS.	19434	03/2013
		MONNAME.	19434	March
		MONTH.	19434	3
		MONYY.	19434	MAR13
		PDJULG. *	19434	2013076F
		PDJULI. *	19434	0100076F
		WEEKDATE.	19434	Sunday, March 17, 2013
		WEEKDAY.	19434	1
		WORDDATE.	19434	March 17, 2013
		WORDDATX.	19434	17 March 2013
	四半期の出力形式	QTR.	19434	1
		QTRR.	19434	I
	時間の出力形式	TIME.	19434	5:23:54
		TIMEAMPM.	19434	5:23:54 AM
		TOD.	19434	05:23:54

タスク	言語要素のタイプ	言語要素	入力	結果
SAS 日付値の書き出し	年の出力形式	YEAR.	19434	2013
		YYMM.	19434	2013M03
		YYMMC.	19434	2013:03
		YYMMD.	19434	2013-03
		YYMMP.	19434	2013.03
		YYMMS.	19434	2013/03
		YYMMN.	19434	201303
		YYMMDD.	19434	13-03-17
		YYMON.	19434	2013MAR
	年/四半期の出力形式	YYQ.	19434	2013Q1
		YYQC.	19434	2013:1
		YYQD.	19434	2013-1
		YYQP.	19434	2013.1
		YYQS.	19434	2013/1
		YYQN.	19434	20131
		YYQR.	19434	2013QI
		YYQRC.	19434	2013:I
		YYQRD.	19434	2013-I
		YYQRP.	19434	2013.I
		YYQRS.	19434	2013/I
		YYQRN.	19434	2013I

* SAS では、ユリウス日付は YYNNN または YYYYNNN 形式で表されます。ここで、YY は 2 桁の年、YYYY は 4 桁の年です。NNN は YY 年または YYYY 年 1 月 1 日からの日数です。SAS は有効な SAS 日付に対応するユリウス日付のみを処理します。

表 7.2 *日時に関するタスク(その 2)*

タスク	言語要素のタイプ	言語要素	入力	結果
日付に関するタスク				
カレンダ形式の日付を SAS 日付値として読み込む 注:YEARCUTOFF=1926	日付の入力形式	DATE.	17MAR13	19434
		DATE9.	17MAR2013	19434

タスク	言語要素のタイプ	言語要素	入力	結果
		DDMMYY.	170313	19434
		DDMMYY8.	17032013	19434
		JULIAN. *	13076	19434
		JULIAN7. *	2013077	19434
		MMDDYY.	031713	19434
		MMDDYY8.	03172013	19434
		MONYY.	MAR13	19418
		YYMMDD.	130317	19434
		YYMMDD8.	20130317	19434
		YYQ.	13q1	19359
		DATETIME	17MAR2013 00:00:00	1679097600
		TIME	14:45:32	53132
現在の日付を SAS 日付値として返す	日付関数	DATE() または TODAY() (同等)	( )	現在の日付の SAS 日付値。
SAS System からカレンダ形式の日付を抽出する	日付関数	DAY	19434	17
		HOUR	19434	5
		JULDATE *	19434	13076
		JULDATE7 *	19434	2013076
		MINUTE	19434	23
		MONTH	19434	3
		QTR	19434	1
		SECOND	19434	54
		WEEKDAY	19434	1
		YEAR	19434	2013
日付を式の定数として書き出す	SAS 日付定数	'*ddmmmyy*'d または '*ddmmmyyyy*'d	'17mar13'd '17mar2013'd	19434

タスク	言語要素のタイプ	言語要素	入力	結果
現在の日付を文字列として書き出す	SYSDATE 自動マクロ変数	SYSDATE	&SYSDATE	SAS が初期化された日付 (DDMMMYY 形式)。
	SYSDATE9	SYSDATE9	&SYSDATE9	SAS が初期化された日付 (DDMMMYYYY 形式)。
時間に関するタスク				
SAS 時間値を書き出す	時間の出力形式	HHMM.	19434	5:24
		HOUR.	19434	5
		MMSS.	19434	323
		TIME.	19434	5:23:54
		TIMEAMPM.	19434	5:23:54 AM
		TOD.	19434	05:23:54
時間を SAS 時間値として読み込む	時間の入力形式	TIME.	5:23:54	19434
現在の時間を文字列として書き出す	SYSTIME 自動マクロ変数	SYSTIME	&SYSTIME	実行時刻 (HH:MM 形式)
現在の時間を SAS 時間値として返す	時間関数	TIME()	( )	実行時の SAS 時間値 (NNNNN.NNN 形式)。
SAS 日時値の時間の部分を返す	時間関数	TIMEPART	17mar2013 5:11:43	5:11:43
日時に関するタスク				
SAS 日時値を書き出す	日時の出力形式	DATEAMPM.	1679097600	17MAR13:12:00:00 AM
		DATETIME	1679097600	17MAR13:00:00:00
日時値を SAS 日時値として読み込む	日時の入力形式	DATETIME	17MAR13:00:00:00	1679097600
現在の日付と時間を SAS 日時値として返す	日時関数	DATETIME()	()	実行時の SAS 日時値 (NNNNNNNNN.N 形式)。
時間間隔に関するタスク				

タスク	言語要素のタイプ	言語要素	入力	結果
2 つの日付または日時の期間中に、指定した時間間隔が生じた回数を返す	時間間隔関数	INTCK	week2 01aug60 01jan13	1368
日付、時間、日時を指定した時間間隔だけ進めた値を返す	時間間隔関数	INTNX	day 17mar12 365	19434

* SAS では、ユリウス日付は YYNNN または YYYYNNN 形式で表されます。ここで、YY は 2 桁の年、YYYY は 4 桁の年です。NNN は YY 年または YYYY 年 1 月 1 日からの日数です。SAS は有効な SAS 日付に対応するユリウス日付のみを処理します。

次の出力形式と入力形式もサポートされています。

- 日付、時間、日時、期間、間隔、タイムゾーンに関する ISO 8601 基本形式および拡張形式。詳細については、"Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations" (*SAS Formats and Informats: Reference*)および"Reading Dates and Times By Using the ISO 8601 Basic and Extended Notations" (*SAS Formats and Informats: Reference*)を参照してください。
- 英語の日付の出力形式と入力形式のうち使用頻度の高いものに相当する各国言語用の出力形式と入力形式。詳細については、"Dictionary of Formats for NLS" (*SAS National Language Support (NLS): Reference Guide*)および"Dictionary of Informats for NLS" (*SAS National Language Support (NLS): Reference Guide*)を参照してください。.

例

例 1:認識可能な日時として、日付値、時間値、日時値を表示する

次の例では、値を日付、時間、日時として表示する方法を示します。SAS 日付値、SAS 時間値、SAS 日時値を、特定の日付、時間、日時形式に変換するための SAS 言語要素を忘れずに指定してください。それぞれの例については、前述の表を参照してください。

注:

- 時間の出力形式では 1 日の秒数をカウントするので、値は 0 - 86400 になります。
- DATETIME.出力形式では、1960 年 1 月 1 日からの秒数をカウントします。日時が 02JAN1960:00:00:01 (整数の 86401)以上の場合、日時値は時間値よりも常に大きいことになります。
- 不適切な値の場合は、データセットの内容を参照して、扱っているデータ値の型を確認してください。

このプログラムでは、DATETIME.、DATE.、TIMEAMPM.出力形式を使用して、86399 という値を日付と時間、カレンダ形式の日付、時間として表示しています。

```
options nodate;
data test;
   Time1=86399;
   format Time1 datetime.;
   Date1=86399;
   format Date1 date9.;
   Time2=86399;
```

```
   format Time2 timeampm.;
run;
proc print data=test;
   title  'Same Number, Different SAS Values';
   footnote1 'Time1 is a SAS DATETIME value';
   footnote2 'Date1 is a SAS DATE value';
   footnote3 'Time2 is a SAS TIME value';
run;
footnote;
```

アウトプット 7.2 *86399 の日時値、日付値、時間値*

Same Number, Different SAS Values

Obs	Time1	Date1	Time2
1	01JAN60:23:59:59	20JUL2196	11:59:59 PM

Time1 is a SAS DATETIME value
Date1 is a SAS DATE value
Time2 is a SAS TIME value

例 2: 日付値の読み込み、書き出し、計算

このプログラムでは、4 回の地区ミーティングの日付を読み込んで、案内状を発送する日付を計算しています。

```
data meeting;

   input region $ mtg : mmddyy8.;
   sendmail=mtg-45;
   datalines;
N  11-24-12
S  12-28-12
E  12-03-12
W  10-04-12
;

proc print data=meeting;
   format mtg sendmail date9.;
   title 'When To Send Announcements';
run;
```

アウトプット 7.3 日付値の計算結果:案内状の発送日

When To Send Announcements

Obs	region	mtg	sendmail
1	N	24NOV2012	10OCT2012
2	S	28DEC2012	13NOV2012
3	E	03DEC2012	19OCT2012
4	W	04OCT2012	20AUG2012

日付と時間の間隔について

定義

期間
: 2 つの日付、時間、日時の差を表す整数です。日付期間は 2 つの SAS 日付の差を日数で表す整数値です。時間期間は 2 つの時間または日時の差を秒数で表す 10 進数値です。

ヒント 日付および日時期間は、大きい日付または日時から小さい日付または日時を引くと簡単に計算できます。SAS 時間を処理している際、期間の始まりと終わりがカレンダの異なる日付にある場合は特に注意が必要です。最も簡単な解決策は、なるべく時間ではなく日時を使用することです。

間隔
: 経過した時間をカウントするための単位です。DAYS、MONTHS、または HOURS などがあります。SAS は、カレンダ、時計、またはその両方の固定点に基づいて日付と時間の間隔を判別します。時間間隔計算の開始点は、デフォルトでは開始値が属している期間の最初です。実際に指定した開始値ではない可能性があります。たとえば、INTCK 関数を使用して 2 つの日付の間の月数を計算する場合、日付と開始値で指定した月の実際の日に関係なく、その月の初日として処理されます。

構文

各種の経過時間を数えるための日付、時間、および日時の間隔が SAS には用意されています。間隔の倍数の作成や、開始点をシフトすることもできます。間隔は、INTCK 関数や INTNX 関数の引数として使用するか、数字付きリストをサポートしているプロシジャ(PLOT プロシジャなど)で使用します。間隔は、次の構文で指定します。

name<multiple><.starting-point>

間隔で使用する用語の定義は次のとおりです。

name
: 間隔キーワードです。間隔キーワードの定義のリストを参照してください。

multiple
間隔の倍数を作成します。*multiple* には任意の正の数を指定できます。デフォルトは 1 です。たとえば、YEAR2 は 2 年の間隔を示します。

.starting␣point
間隔の開始点です。デフォルトの開始点は 1 です。1 より大きい値を指定すると、間隔の範囲内で開始点が後ろにシフトします。シフトの単位は間隔によって異なります。詳細については、次の表を参照してください。たとえば、YEAR.3 は、3 月の始めから翌年の 2 月の終わりまでの 1 年間を示します。

カテゴリ別の間隔

表 7.3 *日時関数で使用される間隔*

カテゴリ	間隔	定義	デフォルトの開始点	シフト期間	例	説明
日付	DAY	日次の間隔	毎日	日	DAY3	日曜日を開始点とする 3 日間隔
	WEEK	週(7 日)	毎日曜日	日(1=日曜日 - 7=土曜日)	WEEK.7	週の先頭を土曜日とした週間隔
	WEEKDAY <*days*W>	金-土-日を含む日次の間隔	毎日	日	WEEKDAY1W	日曜日を週末とした週間隔(稼働日は週 6 日)
		同じ日としてカウント(土日を週末として稼働日 5 日)*days* には週末の日を表す数字を指定します(1=日曜日 ...7=土曜日)。デフォルトでは、*days*=17 になります。			WEEKDAY35W	火曜日と木曜日を週末とした週間隔(稼働日は週 5 日)。W は 3 番目と 5 番目の日が週末であることを示します。
	TENDAY	10 日間隔(米自動車産業の慣例)	毎月の 1 日、11 日、21 日	10 日単位	TENDAY4.2	2 番目の TENDAY 期間を開始点とする 4 回分の TENDAY 期間
	SEMIMONTH	半月間隔	毎月の 1 日、16 日	半月期間	SEMIMONTH2.2	ある月の 16 日から、翌月の 15 日までの期間

カテゴリ	間隔	定義	デフォルトの開始点	シフト期間	例	説明
	MONTH	月間隔	毎月の 1 日	月	MONTH2.2	2 月 - 3 月、4 月 - 5 月、6 月 - 7 月、8 月 - 9 月、10 月 - 11 月、12 月 - 翌年の 1 月
	QTR	四半期(3 か月)間隔	1 月 1 日	月	QTR3.2	4 月 1 日、7 月 1 日、10 月 1 日、1 月 1 日を開始点とする 3 か月
			4 月 1 日			
			7 月 1 日			
			10 月 1 日			
	SEMIYEAR	半年(6 か月)間隔	1 月 1 日	月	SEMIYEAR.3	3 月 - 8 月、9 月 - 2 月の 6 か月
			7 月 1 日			
	YEAR	年間隔	1 月 1 日	月		
日時	任意の日付間隔に DT を追加します	関連する日付間隔に対応する間隔	1960 年 1 月 1 日午前 0 時		DTMONTH	
					DTWEEKDAY	
時間	SECOND	秒間隔	1 日の始まり(午前 0 時)	秒		
	MINUTE	分単位	1 日の始まり(午前 0 時)	分		
	HOUR	時間隔	1 日の始まり(午前 0 時)	時		

例:期間の計算

このプログラムでは、プロジェクトの開始日と終了日を読み込んだ後、プロジェクトの期間を算出します。

```
options nodate pageno=1 linesize=80 pagesize=60;

data projects;
   input Projid @5 startdate date9. @15 enddate date9.;
   Duration=enddate-startdate;
   datalines;
398 17oct1997 02nov1997
942 22jan1998 11mar1998
167 15dec1999 15feb2000
250 04jan2001 11jan2001
;
```

```
proc print data=projects;
   format startdate enddate date9.;
      title 'Days Between Project Start and Project End';
run;
```

アウトプット 7.4 *開始日から終了日までの期間の計算*

Days Between Project Start and Project End

Obs	Projid	startdate	enddate	Duration
1	398	17OCT1997	02NOV1997	16
2	942	22JAN1998	11MAR1998	48
3	167	15DEC1999	15FEB2000	62
4	250	04JAN2001	11JAN2001	7

間隔の境界

日付間隔や時間間隔は、カレンダ上のある特定の開始点と終了点を結びつけるものです。たとえば、MONTH 間隔は、カレンダの特定の月の 1 日から、翌月の 1 日までの期間を表します。30 日間、または 31 日間ということではありません。INTCK 関数や INTNX 関数などで日付間隔や時間間隔を使用すると、カレンダの区分に基づいて計算が行われます。次の表の例を参考にしてください。

表 7.4 *INTCK 関数および INTNX 関数の使用例*

例	結果	説明
`mnthnum1=intck( 'month', '25aug2000'd, '05sep2000'd);`	mnthnum1=1	INTCK 関数で計算されます。MONTH 間隔の数は、指定した期間に月の 1 日目が含まれるかどうかによって異なります。
`mnthnum2=intck( 'month', '01aug2000'd, '31aug2000'd);`	mnthnum2=0	
`next=intnx('month', '25aug2000'd,1);`	next は 01sep2000 を表示	INTNX 関数の結果は、次の間隔の初日に対応する SAS 日付値になります。

注: 開始点が毎年異なる間隔は、WEEK と WEEKDAY だけです。1 年は週の日数 (7)では割り切れないので、日曜日になる日は年ごとに変わります。

単一単位の間隔

1 期間で構成される間隔は、カレンダ上のある開始点を起点として開始されます。次の表を参照してください。

表 7.5 単一単位の間隔

単一単位の間隔	カレンダ上にある開始点
DAY	毎日
WEEKDAY	標準的な平日 • *Start day–End day* • 月曜日 - 月曜日 • 火曜日 - 火曜日 • 水曜日 - 水曜日 • 木曜日 - 木曜日 • 金曜日 - 日曜日
WEEK	毎日曜日
TENDAY	毎月の 1 日、11 日、21 日
SEMIMONTH	毎月の 1 日、16 日
MONTH	毎月の 1 日目
QTR	1 月 1 日、4 月 1 日、7 月 1 日、10 月 1 日
SEMIYEAR	1 月 1 日、7 月 1 日
YEAR	1 月 1 日

1 期間で構成される時間間隔については、次の表を参照してください。

表 7.6 単一単位の時間間隔

単一単位の時間間隔	開始点
SECOND	毎秒
MINUTE	毎分
HOUR	毎時

複数単位の間隔

複数週の間隔以外の複数単位の間隔

MONTH2 や DAY50 など、複数の単位で構成される間隔もカレンダに依存しますが、考慮すべき点がさらに 1 つあります。SAS では、期間の先頭(月の初日など)は認識できますが、その期間が間隔のどこに位置しているのかは認識できません。たとえば、2 か月の間隔で、10 月 1 日がどちらの月を表しているのかは特定できません。

複数の単位で構成される間隔は、週単位のものを除き、すべて 1960 年 1 月 1 日を開始点として作成されます。この日を起点として日付が計算され、カレンダ上での間隔の開始点が決定します。実際にプログラムを作成するにあたっては、1 年を等分できる間隔の場合は、現在の年から開始させるようにします。たとえば、MONTH2 の間隔は、1 月、3 月、5 月、7 月、9 月、11 月に開始させます。次の例を考えてみます。

表 7.7 *Month2 の間隔*

SAS ステートメント	結果
`howmany1=intck('month2','15feb2000'd, '15mar2000'd);`	howmany1=1
`count=intck('day50','01oct1998'd, '01jan1999'd);`	count=1

この例では、1960 年 1 月 1 日を開始点とする 50 日間、次の 50 日間、というように計算しています。この計算が実行されると、1998 年 10 月 1 日 - 1999 年 1 月 1 日を 1 つの DAY50 間隔として計算します。次の例では、INTNX 関数を使用して、DAY50 間隔の次の開始日を調べています。

表 7.8 *INTNX 関数の使用*

SAS ステートメント	結果
`start=intnx('day50','01oct98'd,1);`	SAS 日付値 14200 (1998 年 11 月 17 日)

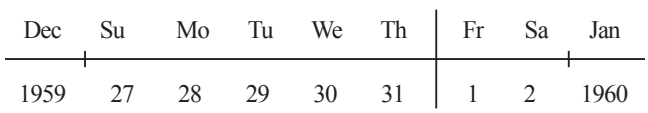
次の間隔は 1998 年 11 月 17 日に始まります。

時間の間隔は、1 日の中での期間を表します。時間間隔の開始点は、1 日の始まり、つまり午前 0 時になります。たとえば、HOUR8 という間隔では、1 日を 00:00 - 08:00、8:00 - 16:00、16:00 - 24:00 (翌日の午前 0 時)に分けます。

複数週の間隔

WEEK2 など、複数の週の場合の間隔は特殊な間隔です。週を単位とする間隔は、通常は日曜日に始まり、週は日曜日が過ぎるたびに 1 つ計算されます。複数の週にまたがる間隔は、1960 年 1 月 1 日を起点として計算することはできません。この日は、次の図に示すように金曜日に当たるためです。

図 7.2 *複数週の間隔の計算*



したがって、1 番目の週間隔は、1960 年 1 月 1 日が含まれる週の日曜日、つまり 1959 年 12 月 27 日に始まります。複数の週で構成される間隔は、この日を起点として計算されます。次の例では、1998 年の 8 月に含まれる、2 週単位の間隔の数を計算しています。

表 7.9 2 週単位の間隔数の計算

SAS ステートメント:	結果
`count=intck('week2','01aug98'D, '31aug98'D);`	count=3

次の 2 週単位間隔の開始日を調べるには、次のように INTNX 関数を使用します。

表 7.10 INTNX 関数による間隔開始日の特定

SAS ステートメント:	結果
`begin=intnx('week2','01aug1998'd,1);`	begin は SAS 日付値 14093 (1998 年 8 月 2 日)になります。

次の間隔は 8 月 16 日に始まります。

間隔のシフト

間隔のシフトの使用

間隔を、処理対象のデータ内の期間と一致させたいときは、データ間隔の開始点をシフトすると便利です。たとえば、会社の会計年度の開始日が 7 月 1 日である場合は、YEAR.7 という間隔を指定すると、7 月に開始する年度を作成できます。同じように、YEAR4.11 という間隔を指定することで、米国の大統領選挙の期間と一致する期間を作成できます。ここでは、間隔のシフトの使用方法と、間隔のシフトのしくみについて説明します。

間隔のシフトの使用方法

下位単位を基準にして間隔をシフトする場合、指定するシフト値は、間隔に含まれる下位単位の合計数以下にする必要があります。たとえば、YEAR.12 は 12 月を始点とする 1 年間であり、有効な間隔ですが、YEAR.13 は無効な間隔です。同様に、YEAR2.25 という間隔も無効です。2 年という間隔には、25 番目の月がないためです。

また、間隔そのものをシフトすることはできません。たとえば、MONTH という間隔はシフトできません。MONTH をシフトしようとするときの下位単位は 1 か月ですが、MONTH には月という下位単位が 1 つ分しか含まれていないためです。ただし、複数の単位で構成される間隔を、下位単位を基準としてシフトすることはできます。たとえば、MONTH2.2 と指定すると、2 番目の月の 1 日目を開始日とする 2 か月間になります。

間隔のシフトのしくみ

SAS では、間隔はすべて 1960 年 1 月 1 日を起点として作成されます(週単位の間隔は除く)。この起点は、指定された下位単位の数だけ暦の先へと移動します。間隔のシフトは、その移動後の地点から計算されます。たとえば、DAY50.5 という間隔のシフトを作成するとします。まず、1960 年 1 月 1 日を 1 日目とする 50 日の間隔が作成されます。次に、5 日目に移動します。移動による差つまり移動量は、4 日間です。間隔のシフトは、この 5 日目からカウントされます。次に示す INTNX 関数の例では、次の間隔は 1960 年 1 月 5 日に開始されます。

表 7.11 INTNX 関数による間隔開始点の特定

SAS ステートメント:	結果
`start=intnx('day50.5','01jan1960'd,1);`	SAS 日付値 4 (1960 年 1 月 5 日)

週単位の間隔をシフトする場合は、まず、1960 年 1 月 1 日が含まれる週の日曜日、つまり 1959 年 12 月 27 日を起点として間隔が作成されます。この起点は、指定された下位単位の数だけ暦の先へと移動します。たとえば、WEEK2.8 という間隔(期間内の 2 番目の日曜日を起点とする 2 週間)を作成するとします。その場合は、1960 年 1 月 1 日が含まれる週の日曜日を起点とした、2 週間の間隔が作成されます。間隔のシフトの計算は、その間隔の 8 日目から開始されます。INTNX 関数を使用すると、間隔の次の開始点を示すことができます。

表 7.12 INTNX 関数を使用して次の間隔の開始点を示す例

SAS ステートメント:	結果
`start=intnx('week2.8','01jan1960'd,1);`	SAS 日付値 2 (1960 年 1 月 3 日)

時間値の間隔をシフトすることもできます。たとえば、HOUR8.7 という間隔は、1 日を 06:00 - 14:00、14:00 - 22:00、22:00 - 06:00 に分けます。

カスタム間隔

INTERVALDS=システムオプションを使用する際に、カスタム間隔を定義し、間隔データセットを新しい間隔名に関連付けることができます。間隔名には、予約 SAS 名は使用できません。その間隔の日付は、作成した SAS データセットに格納されます。データセットには変数 Begin を含める必要があります。各オブザベーションで、Begin 変数は間隔の開始を表します。間隔の終了を表すには 2 番目の変数 End を指定しますが、これは必須ではありません。データセットに End 変数が存在しない場合、間隔の終了は、次の Begin 変数値から推測されます。カスタム間隔を定義すると、標準の間隔と同じように INTCK 関数と INTNX 関数で使用できます。

INTERVALDS=システムオプションでは、許容可能な間隔の数を増やすことができます。間隔の標準リスト(DAY、WEEKDAY など)に加え、INTERVALDS=にリストされている名前も有効です。

注: ネストされたカスタム間隔はサポートされません。

販売カレンダの間隔:ISO 8601 準拠

小売業界では、4-4-5、4-5-4、5-4-4 のいずれかの形式に基づき、1 年のカレンダを 13 週間からなる 4 つの期間に分けてデータを計算することがよくあります。1 番目、2 番目、3 番目の数字は、各期間の 1 番目、2 番目、3 番目の月の週の数を示します。小売カレンダの間隔を使用すると、週の定義が年度ごとに一定なので、各年の比較が簡単になります。

この形式から作成される間隔は、INTCINDEX、INTCK、INTCYCLE、INTFIT、INTFMT、INTGET、INTINDEX、INTNX、INTSEAS、INTSHIFT のいずれの関数でも使用できます。

次の表に、小売業界で使用され、ISO 8601 を遵守しているカレンダの間隔を示します。

表 7.13 *小売業界で使用されているカレンダの間隔*

間隔	説明
YEARV	ISO 8601 の年次間隔を指定します。ISO 8601 の年は、月曜日または 1 月 4 日の直前に始まります。ただし、前年の 12 月に始まることもあるので注意してください。ISO 8601 の年はうるう週を含む場合があります。開始の下位単位 *s* が、ISO 8601 の週(WEEKV).に書き込まれます。
R445YR	YEARV と同じですが、小売業界では、開始の下位単位 *s* が 4-4-5 の月(R445MON).です。
R454YR	YEARV と同じですが、小売業界では、開始の下位単位 *s* が 4-5-4 の月(R454MON).です。
R544YR	YEARV と同じですが、小売業界では、開始の下位単位 *s* が 5-4-4 の月(R544MON).です。
R445QTR	小売業界の 4-4-5 四半期単位の間隔を指定します(ISO 8601 の 13 週単位)。第 4 四半期はうるう週を含む場合があります。開始の下位単位 *s* は 4-4-5 の月(R445MON)です。
R454QTR	小売業界の 4-5-4 四半期単位の間隔を指定します(ISO 8601 の 13 週単位)。第 4 四半期はうるう週を含む場合があります。開始の下位単位 *s* は 4-5-4 の月(R454MON)です。
R544QTR	小売業界の 5-4-4 四半期単位の間隔を指定します(ISO 8601 の 13 週単位)。第 4 四半期はうるう週を含む場合があります。開始の下位単位 *s* は 5-4-4 の月(R544MON)です。
R445MON	小売業界の 4-4-5 の月単位の間隔を指定します。3 か月目、6 か月目、9 か月目、12 か月目が ISO 8601 の 5 週間です。ただし、12 か月目はうるう週を含むことがあります。それ以外のすべての月は ISO 8601 の 4 週間です。R445MON の間隔は ISO 年の 1 週目、5 週目、9 週目、14 週目、18 週目、22 週目、27 週目、31 週目、35 週目、40 週目、44 週目、48 週目です。開始の下位単位 *s* は 4-4-5 の月(R445MON)です。
R454MON	小売業界の 4-5-4 の月単位の間隔を指定します。2 か月目、5 か月目、8 か月目、11 か月目は ISO 8601 の 5 週間です。ただし、12 か月目はうるう週を含むことがあります。R454MON の間隔は、ISO 年の 1 週目、5 週目、10 週目、14 週目、18 週目、23 週目、27 週目、31 週目、36 週目、40 週目、44 週目、49 週目です。開始の下位単位 *s* は 4-5-4 の月(R454MON)です。
R544MON	小売業界の 5-4-4 の月単位の間隔を指定します。1 か月目、4 か月目、7 か月目、10 か月目は ISO 8601 の 5 週間です。それ以外のすべての月は ISO 8601 の 4 週間です。ただし、12 か月目はうるう週を含む場合があります。R544MON の間隔は、ISO 年の 1 週目、6 週目、10 週目、14 週目、19 週目、23 週目、27 週目、32 週目、36 週目、40 週目、45 週目、49 週目です。開始の下位単位 *s* は 5-4-4 の月(R544MON)です。

間隔	説明
WEEKV	ISO 8601 の週単位の間隔(7 日間)を指定します。各週は月曜日から始まります。開始の下位単位 *s* は日(DAY)で計算されます。WEEKV は、WEEKV.1 が月曜日から始まり、WEEKV.2 が火曜日で始まるという点で WEEK とは異なります。

8 章

エラー処理とデバッグ

SAS のエラーの種類

SAS が認識するエラーの種類の要約

エラー処理は、SAS 処理のコンパイルフェーズと実行フェーズで行われます。SAS プログラムのデバッグは、SAS ログに表示されたメッセージを理解し、プログラムを修正することによって行います。また、DATA ステップデバッガを使用すると、DATA ステップの論理エラーを実行時に検出することができます。

SAS System が検出するエラーには、次の 5 種類があります。

表 8.1 *エラーの種類*

エラーの種類	エラーが発生した場合	エラーが検出された場合
構文	プログラムステートメントが SAS 言語の規則に従っていない場合	コンパイル時

エラーの種類	エラーが発生した場合	エラーが検出された場合
セマンティック	SAS ステートメントの構文の形式は正しいものの、その要素が有効ではない使い方をしている場合	コンパイル時または実行時
実行時	実行したプログラムが異常終了した場合	実行時
データ	データ値が無効な場合	実行時
マクロ関連	マクロ機能の使用方法が正しくない場合	マクロ、DATA ステップ、PROC ステップのコンパイル時あるいは実行時

構文エラー

構文エラーは、プログラムステートメントが SAS 言語の構文規則に従っていない場合に発生します。構文エラーの例を次に示します。

- スペルミスがある SAS キーワード
- 引用符の不一致
- セミコロンがない
- 無効なステートメントオプション
- 無効なデータセットオプション

構文エラーが検出されると、エラー回復機能により、その部分が何を意図しているのかがまず分析されます。その分析に基づいてエラーの修正が試みられます。修正できたと判断された場合は、プログラムの処理が続行されます。エラーを修正できなかった場合は、SAS ログにエラーメッセージが出力されます。構文エラーを修正しない場合、NOAUTOCORRECT システムオプションを設定できます。詳細については、*SAS System Options: Reference* で AUTOCORRECT システムオプションを参照してください。

次の例では、DATA ステートメントにスペルミスがあるため、SAS ログに警告メッセージが出力されます。スペルミスのあるキーワードの解釈が可能だったため、プログラムは実行され、出力も生成されます。

```
date temp;
   x=1;
run;

proc print data=temp;
run;
```

ログ 8.1 *SAS ログ:構文エラー(スペルミスのあるキーワード)*

```
39 date temp; ---- 14 WARNING 14-169:Assuming the symbol DATA was misspelled as
date.40      x=1; 41   run; NOTE:The data set WORK.TEMP has 1 observations and 1
variables.NOTE:DATA statement used (Total process time): real time 0.00 seconds
cpu time 0.00 seconds 42 43 proc print data=temp; 44 run; NOTE:There were 1
observations read from the data set WORK.TEMP.NOTE:PROCEDURE PRINT used (Total
process time): real time 0.00 seconds cpu time 0.00 seconds 45 proc printto; run;
```

SAS ログに表示されたメッセージで十分に理解できるエラーもありますが、エラーメッセージとの対応が付けにくいエラーもあります。これは、SAS System がエラーの発生箇所を特定できない場合があるためです。たとえば、SAS ステートメントの末尾にセミコロン(;)を付け忘れたとします。SAS System は、この場合、エラーの発生した箇所をいつも正確に検出するとは限りません。SAS ステートメントは、任意の位置で開始および終了できるためです。次の例では、DATA ステートメントの末尾にセミコロン(;)が付いていません。SAS ログには、ERROR という語に続いて、エラーが発生したと思われる箇所が示され、エラーの説明が出力されます。DATA ステップの処理は自動的に停止します。

```
data temp
   x=1;
run;

proc print data=temp;
run;
```

ログ 8.2 *SAS ログ:構文エラー(セミコロンの欠損)*

```
67 data temp 68 x=1; - 22 76 ERROR 22-322:Syntax error, expecting one of the
following: a name, a quoted string, (, /, ;, _DATA_, _LAST_, _NULL_.ERROR
76-322:Syntax error, statement will be ignored.69   run; NOTE:The SAS System
stopped processing this step because of errors.NOTE:DATA statement used (Total
process time): real time 0.01 seconds cpu time 0.01 seconds 70 71 proc print
data=temp; 72 run; NOTE:There were 1 observations read from the data set
WORK.TEMP.NOTE:PROCEDURE PRINT used (Total process time): real time 0.00 seconds
cpu time 0.00 seconds 73 proc printto; run;
```

後続のステップが実行されるかどうかは、SAS System の実行方法や、オペレーティングシステムによって異なります。

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できます。

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

セマンティックエラー

セマンティックエラーは、SAS ステートメントの構文の形式は正しいものの、その構文に含まれる要素が有効ではない使い方をしている場合に発生します。セマンティックエラーはコンパイル時に検出され、SAS System は構文チェックモードに切り替わります。(構文チェックモードの詳細については、“構文チェックモード” (134 ページ)を参照してください)。

セマンティックエラーの例には次のようなものがあります。

- 関数の引数の数が間違っている場合。
- 文字変数が必要な箇所に、数値変数を指定している場合。
- 配列への参照が無効な場合。
- 変数が初期化されていない場合。

次の例では、配列 All への参照が無効であることがコンパイル時に検出されます。

```
data _null_;
   array all{*} x1-x5;
   all=3;
```

```
   datalines;
1 1.5
. 3
2 4.5
3 2 7
3 . .
;

run;
```

ログ 8.3 *SAS ログ:セマンティックエラー(配列への参照が無効な場合)*

```
81   data _null_; 82      array all{*} x1-x5; ERROR: invalid reference to the
array all.83      all=3; 84      datalines; NOTE:The SAS System stopped
processing this step because of errors.NOTE:DATA statement used (Total process
time): real time 0.15 seconds cpu time 0.01 seconds 90 ; 91 92 run; 93 proc
printto; run;
```

次の例も、コンパイル時に発生するセマンティックエラーの一種です。この DATA ステップでは、ライブラリ参照名 Somelib が、あらかじめ LIBNAME ステートメントを使って割り当てられていないためにエラーとなります。

```
data test;
   set somelib.old;
run;
```

ログ 8.4 *SAS ログ:セマンティックエラー(ライブラリ参照が割り当てられていない場合)*

```
101 data test; ERROR:Libname SomeLib is not assigned.102     set somelib.old;
103  run; NOTE:The SAS System stopped processing this step because of
errors.WARNING:The data set WORK.TEST may be incomplete.When this step was
stopped there were 0 observations and 0 variables.
```

実行時に発生するセマンティックエラーの例としては、次の NOTE メッセージが出力される場合などがあります。"NOTE:SAS went to a new line when input statement reached past the end of a line."この NOTE は、FLOWOVER を使用しているときに、INPUT ステートメントのすべての変数が完全には読み取れない場合に SAS ログに書き出されます。

別のセマンティックエラーとしては、初期化されていない変数の検出があります。デフォルトでは、エラーにはレポートされませんが、SAS ログに NOTE が書き出されます。変数が初期化されておらず、システムオプション VARINITCHK=ERROR である場合、DATA ステップの処理は停止し、SAS ログにエラーメッセージが書き出されます。

実行時エラー

定義

実行時エラーは、データ値を処理するプログラムの実行時に起こったエラーです。ほとんどの場合、実行時エラーが発生すると SAS ログに警告メッセージや NOTE が表示されますが、プログラムの実行は継続されます。[1] 実行時エラーの発生箇所は、通常、NOTE またはエラー(ERROR)メッセージの中に行番号とカラム番号で示されます。

主な実行時エラーの原因としては、次のようなものがあります。

1 SAS System を非対話型モードで実行している場合は、重大なエラーが発生すると、SAS System は構文チェックモードに切り替わり、プログラムの処理は停止する場合があります。

- 関数の引数が無効な場合。
- 算術演算が無効な場合(0 による除算など)。
- BY グループ処理のオブザベーションの順序が適切でない場合。
- 配列の添字が範囲外にある場合など、参照している配列のメンバが存在しない場合。
- INFILE ステートメントまたは FILE ステートメントで、SAS データセットやその他のファイルを開いたり閉じたりできない場合。
- INPUT ステートメントがデータ行と一致していない場合(INPUT ステートメントに指定した変数のカラムが間違っている場合や、文字変数を文字変数として指定していない場合など)。

リソース不足状態

実行時エラーは、リソース不足状態に陥った場合にも発生します。リソース不足状態とは、ディスクの空き容量やメモリが不足したことによって、処理を完了できない状態のことをいいます。リソース不足状態に陥ると、SAS System は差し当たり必要なだけのリソースを確保しようとします。たとえば、リソース不足の場合に次のアクションを実行する許可を要求します。

- 利用されなくなった一時データセットを削除する。
- マクロ変数が格納されているメモリを解放する。

ウィンドウ環境でリソース不足状態に陥った場合、CLEANUP システムオプションを使用すると、リクエスタパネルを表示できます。このリクエスタパネルを使用して、エラーの解決方法を選択できます。SAS System をバッチモード、非対話型モード、対話型ラインモードで実行している場合、CLEANUP システムオプションの動作はオペレーティングシステムによって異なります。詳細については、*SAS System Options: Reference* の CLEANUP システムオプション、および使用している動作環境に対応する SAS ドキュメントを参照してください。

例

次の例では、2 番目のオブザベーションのデータ値を使用して、割り当てステートメント内で除算を実行した時点で実行時エラーが発生します。算術演算では 0 による除算が無効であるために、実行時エラーが発生します。

```
data inventory;
   input Item $ 1-14 TotalCost 15-20
         UnitsOnHand 21-23;
   UnitCost=TotalCost/UnitsOnHand;
   datalines;
Hammers       440   55
Nylon cord    35    0
Ceiling fans  1155  30
;

proc print data=inventory;
   format TotalCost dollar8.2 UnitCost dollar8.2;
run;
```

ログ 8.5 *SAS ログ:実行時エラー(0 による除算)*

```
115 data inventory; 116 input Item $ 1-14 TotalCost 15-20 117 UnitsOnHand 21-23;
118 UnitCost=TotalCost/UnitsOnHand; 119 datalines; NOTE:Division by zero
detected at line 118 column 22.RULE:      ----+----1----+----2----+----3----
+----4----+----5--- 121        Nylon cord    35    0 Item=Nylon cord
TotalCost=35 UnitsOnHand=0 UnitCost=._ERROR_=1 _N_=2 NOTE:Mathematical
operations could not be performed at the following places.The results of the
operations have been set to missing values.Each place is given by:(Number of
times) at (Line):(Column).1 at 118:22 NOTE:The data set WORK.INVENTORY has 3
observations and 4 variables.NOTE:DATA statement used (Total process time): real
time 0.03 seconds cpu time 0.00 seconds 123 ; 124 125 proc print data=inventory;
126 format TotalCost dollar8.2 UnitCost dollar8.2; 127 run; NOTE:Writing HTML
Body file:sashtml1.htm NOTE:There were 3 observations read from the data set
WORK.INVENTORY.NOTE:PROCEDURE PRINT used (Total process time): real time 0.56
seconds cpu time 0.01 seconds
```

アウトプット 8.1 *SAS 出力:実行時エラー(0 による除算)*

The SAS System

Obs	Item	TotalCost	UnitsOnHand	UnitCost
1	Hammers	$440.00	55	$8.00
2	Nylon cord	$35.00	0	.
3	Ceiling fans	$1155.00	30	$38.50

この例では、ステップ全体が実行され、出力結果にある変数 UnitCost に欠損値が割り当てられるとともに、SAS ログに次の項目が表示されます。

- エラーを記述するメッセージ
- 入力バッファに格納された値
- エラー発生時のプログラムデータベクトルの内容
- エラーを説明する NOTE メッセージ

なお、プログラムデータベクトル中の値には、自動変数_N_および_ERROR_も含まれています。これらの自動変数は、一時的に割り当てられるものであり、データセットと一緒には格納されません。

次の実行時エラーの例では、配列の処理中に、範囲外の配列添字の値が検出されます。このため、SAS ログにエラーメッセージが表示され、処理は停止します。

```
data test;
   array all{*} x1-x3;
   input I measure;
   if measure > 0 then
      all{I} = measure;
   datalines;
1 1.5
. 3
2 4.5
;

proc print data=test;
run;
```

ログ 8.6 *SAS ログ:実行時エラー(添字が範囲外)*

```
163 data test; 164 array all{*} x1-x3; 165 input I measure; 166 if measure > 0
then 167 all{I} = measure; 168 datalines; ERROR:Array subscript out of range at
line 167 column 7.RULE:      ----+----1----+----2----+----3----+----4----
+----5--- 170        .3 x1=. x2=. x3=.I=. measure=3 _ERROR_=1 _N_=2 NOTE:The SAS
System stopped processing this step because of errors.WARNING:The data set
WORK.TEST may be incomplete.When this step was stopped there were 1 observations
and 5 variables.WARNING:Data set WORK.TEST was not replaced because this step
was stopped.NOTE:DATA statement used (Total process time): real time 0.00
seconds cpu time 0.00 seconds 172 ; 173 174 proc print data=test; 175 run;
NOTE:No variables in data set WORK.TEST.NOTE:PROCEDURE PRINT used (Total process
time): real time 0.00 seconds cpu time 0.00 seconds 176 proc printto; run;
```

データエラー

定義

データエラーは、データ値が無効で、プログラム内に記述した SAS ステートメントに適切でない場合に発生します。たとえば、変数を数値として定義している場合に、実際のデータ値が文字であったときに発生します。データエラーは、プログラムの実行時に検出されますが、プログラムの実行は継続され、次の処理が行われます。

- 不正なデータの NOTE を SAS ログに表示します。
- 無効な値が含まれている入力行とカラム番号を、SAS ログに表示します。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。
- 罫線の下にオブザベーションが出力されます。
- 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。

次の例では、変数 Number に文字値が入力されたことが原因となって、プログラムの実行中にデータエラーが発生します。

```
data age;
   input Name $ Number;
   datalines;
Sue 35
Joe xx
Steve 22
;

proc print data=age;
run;
```

SAS ログには、プログラムの 8 行目の 5 - 6 カラム目でエラーが発生していることが表示されます。

ログ 8.7 *SAS ログ:データエラー*

```
234 data age; 235 input Name $ Number; 236 datalines; NOTE:Invalid data for
Number in line 238 5-6.RULE:      ----+----1----+----2----+----3----+----4----
+----5--- 238        Joe xx Name=Joe Number=._ERROR_=1 _N_=2 NOTE:The data set
WORK.AGE has 3 observations and 2 variables.NOTE:DATA statement used (Total
process time): real time 0.01 seconds cpu time 0.00 seconds 240 ; 241 242 proc
print data=age; 243 run; NOTE:Writing HTML Body file:sashtml2.htm NOTE:There
were 3 observations read from the data set WORK.AGE.NOTE:PROCEDURE PRINT used
(Total process time): real time 0.07 seconds cpu time 0.04 seconds
```

アウトプット 8.2 *SAS 出力:データエラー*

The SAS System

Obs	Name	Number
1	Sue	35
2	Joe	.
3	Steve	22

INVALIDDATA=システムオプションを使用して、プログラム実行時に無効な値が検出された場合に、変数に値を割り当てることもできます。詳細については、INVALIDDATA=システムオプション(*SAS System Options: Reference*)を参照してください。

エラーレポートのフォーマット修飾子

INPUT ステートメントでは、フォーマット修飾子?および??をエラーレポートに使用できます。これらのフォーマット修飾子を利用することで、SAS ログに書き出す情報の量を制御できます。修飾子?および??は、どちらも無効なデータメッセージを抑制するために使用します。ただし、??修飾子を使用すると、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;
  ```

どちらの場合も、変数 X の値が無効なときは欠損値に設定されます。

マクロ関連エラー

マクロ関連のエラーには、次の 2 種類があります。

- マクロ機能の使用時に生成される、マクロのコンパイル時および実行時のエラー
- マクロ機能によって作成された SAS プログラムのエラー

マクロの詳細については、*SAS Macro Language: Reference* を参照してください。

SAS のエラー処理

構文チェックモード

構文チェックモードの概要

DATA ステップのステートメントに構文エラーがあるときに処理を停止する場合、構文チェックモードを有効にすることができます。これには、バッチモードまたは非対話型モードで SYNTAXCHECK システムオプションを設定するか、ウィンドウ環境で DMSSYNCHK システムオプション設定します。

プログラムがデータセットを作成する場合にのみ、構文チェックモードに入ることができます。DATA _NULL_ステートメントを使用する場合、データセットが作成されないの
  ```

で、構文チェックモードに入ることはできません。この場合、SYNTAXCHECK または DMSSYNCHK システムオプションを使用しても無効です。

構文チェックモードでは、OBS=オプションが内部で 0 に設定され、REPLACE/ NOREPLACE オプションが NOREPLACE に設定されます。このオプションが有効な場合、SAS は次の動作を実行します。

- DATA ステップまたは PROC ステップで残りのステートメントが読み込まれます。
- プログラムステートメントが有効なステートメントかどうかがチェックされます。
- グローバルステートメントが実行されます。
- SAS ログにエラーメッセージを書き出します。
- プログラムステートメント内で指定されている出力データセットのディスクリプタ部が作成されます。
- 作成された新しいデータセットには、オブザベーションが書き出されません。
- DATASETS プロシジャや CONTENTS プロシジャなどの例外を除いて、後続の DATA ステップまたはプロシジャは原則として実行されません。

注: 構文チェックモードに切り替わった後は、既存のデータセットと同じ名前を持つデータセットが作成されても、古いデータセットは上書きされません。

構文チェックモードが有効になっている場合、SAS はエラーの発生箇所を強調し、エラーを番号によって識別します。次に、構文チェックモードに切り替わり、プログラムが終了するまでこのモードで実行されます。構文チェックモードでは、すべての DATA ステップステートメントと PROC ステップステートメントが有効になります。

構文チェックモードの有効化

SYNTAXCHECK システムオプションを使用すると、非対話型モードまたはバッチモードで SAS を実行しているときに構文チェックモードを有効にすることができます。DMSSYNCHK システムオプションを使用すると、ウィンドウ環境で SAS を実行しているときに構文チェックモードを有効にすることができます。プログラムがデータセットを作成する場合にのみ、これらのシステムオプションを使用できます。DATA _NULL_ステートメントを使用している場合、これらのオプションが無視されます。

構文チェックモードを無効にするには、NOSYNTAXCHECK システムオプションと NODMSSYNCHK システムオプションを使用します。

OPTIONS ステートメントでは、SYNTAXCHECK または DMSSYNCHK を有効にする OPTIONS ステートメントの後で、適用するステップを配置します。ステップ内に OPTIONS ステートメントを配置すると、次のステップの開始部分まで SYNTAXCHECK または DMSSYNCHK は有効になりません。

これらのシステムオプションの詳細については、“DMSSYNCHK System Option” (*SAS System Options: Reference*)と “SYNTAXCHECK System Option” (*SAS System Options: Reference*)を参照してください。

複数のエラーの処理

SAS System がプログラムの処理を中止するか、それともエラーフラグを設定して処理を継続するかは、エラーの種類と重大度、SAS System の実行方法、動作環境によって異なります。特定の種類のエラーについては、検出された後も、プロシジャ内の個々のステートメントのチェックを継続します。場合によっては、SAS は、単一のステートメントで複数のエラーを検出し、所定の状況でより多くのエラーメッセージを発行できます。これは、エラーを含んでいるステートメントが出力 SAS データセットを作成した場合によく起こります。

次に示すのは、エラーが 2 個あるステートメントの例です。

```
data temporary;
   Item1=4;
run;

proc print data=temporary;
   var Item1 Item2 Item3;
run;
```

ログ 8.8 *SAS ログ:複数のプログラムエラー*

```
273 data temporary; 274 Item1=4; 275 run; NOTE:The data set WORK.TEMPORARY has 1
observations and 1 variables.NOTE:DATA statement used (Total process time): real
time 0.01 seconds cpu time 0.01 seconds 276 277 proc print data=temporary;
ERROR:Variable ITEM2 not found.ERROR:Variable ITEM3 not found.278     var Item1
Item2 Item3; 279  run; NOTE:The SAS System stopped processing this step because
of errors.NOTE:PROCEDURE PRINT used (Total process time): real time 0.52 seconds
cpu time 0.00 seconds 280 proc printto; run;
```

SAS ログには、変数 Item2 と変数 Item3 に対して、それぞれ 1 つずつエラーメッセージが表示されています。

エラーの検出される可能性がほとんどない、デバッグ済みのプログラムを実行するときに、エラーを 1 つ発生させてプログラムを強制的に異常終了したい場合があります。これには ERRORABEND システムオプションを使用します。

チェックポイントモードと再起動モード

チェックポイントモードおよび再起動モードの概要

チェックポイントモードと再起動モードを一緒に使用すると、完了前に終了するバッチプログラムを、ラベリングされたコードセクションから再度サブミットできる環境が構築されます。実行は、障害発生時に実行されていた DATA や PROC ステップまたはラベリングされたコードセクションで再開されます。

ラベリングされたコードセクションは、DATA または PROC ステップの外側にある *label:* で始まり、DATA または PROC ステップの外側にある次の *label:* の前の RUN ステートメントで終わる SAS コードです。これらのラベルは一意でなければなりません。一方のデータがもう一方に依存しているためにグループ化する必要がある DATA または PROC ステップをグループ化する際に、ラベリングされたコードセクションの使用を検討してください。

次の例には、ラベリングされたコードセクションが 2 つあります。1 番目のラベリングされたコードセクションは、**readSortData:**というラベルで始まり、**proc sort data=mylib.mydata;**の **run;**ステートメントで終わります。2 番目のラベリングされたコードセクションは、**report:**というラベルで始まり、**proc report data=mylib.mydata;**の **run;**ステートメントで終わります。

```
readSortData:
data mylib.mydata;
...more sas code...
run;

proc sort data=mylib.mydata;
...more sas code...
run;

report:
proc report data=mylib.mydata;
```

```
...more sas code...;
run;
endReadSortReport:
```

注: チェックポイントモードと再起動モードで *label:*の使用が有効になるのは、DATA または PROC ステートメントの外側のみです。ラベリングされたコードセクションの場合、チェックポイントモードと再起動モードは、DATA ステップまたはマクロ内のラベルでは無効です。

チェックポイントモードと再起動モードは、DATA ステップや PROC ステップの場合、またはラベリングされたコードセクションの場合のどちらかで有効ですが、両方で同時に使用することはできません。ステップバイステップでチェックポイントモードおよび再起動モードを使用するには、ステップチェックポイントモードまたはステップ再起動モードを使用します。コードセクションのグループに基づいてチェックポイントモードおよび再起動モードを使用するには、ラベルチェックポイントモードまたはラベル再起動モードを使用します。コードの各グループは一意のラベルで識別されます。ラベルを使用する場合、SAS プログラムのすべてのステップはラベリングされたコードセクションに属している必要があります。

チェックポイントモードが有効な場合、SAS は DATA ステップと PROC ステップに関する情報をチェックポイントライブラリに記録します。バッチプログラムが前もって終了した場合、再起動モードでプログラムを再度サブミットすることで実行を完了できます。再起動モードで、グローバルステートメントが再実行され、マクロ定義が再コンパイルされ、マクロが再実行されます。SAS は、チェックポイントライブラリのデータを読み込み、どのステップまたはラベリングされたコードセクションを完了するかを判別します。プログラム実行は、障害発生時に実行されていたステップまたはラベルで再開されます。

チェックポイント再起動データは、完了した DATA ステップと PROC ステップまたはラベリングされたコードセクション、あるいは完了していないステップまたはラベリングされたコードセクションに関する情報のみを格納します。チェックポイント再起動データは次の情報を格納しません。

- マクロ変数とマクロ定義に関する情報
- SAS データセットに関する情報
- 完了していないステップまたはラベリングされたコードセクションで処理された可能性のある情報

注: チェックポイントモードは、コマンドを SAS にサブミットするための DM ステートメントを格納しているバッチプログラムで無効です。チェックポイントモードが有効な場合、SAS が DM ステートメントを検出すると、チェックポイントモードが無効になり、チェックポイントカタログエントリが削除されます。

ベストプラクティスとして、ラベリングされたコードセクションを使用する場合、プログラムの最後にラベルを追加します。プログラムが正常に完了すると、ラベルはチェックポイント再起動データに記録されます。プログラムを再起動モードでサブミットし直すと、SAS は、プログラムが正常に完了済みであることを認識します。

DATA ステップまたは PROC ステップを再実行する必要がある場合、ステップの直前にグローバルステートメント CHECKPOINT EXECUTE_ALWAYS を追加できます。このステートメントは、チェックポイント再起動データを考慮せずに次のステップを必ず実行するように SAS に指示します。これはステートメントに続くステップにのみ適用可能です。詳細については、“CHECKPOINT EXECUTE_ALWAYS Statement” (*SAS Statements: Reference*)を参照してください。

DATA ステップと PROC ステップでチェックポイントモードと再起動モードを有効にするには、SAS でバッチプログラムを起動する際にシステムオプションを使用します。

- STEPCHKPT システムオプションでは、チェックポイントモードを有効にできます。これにより、チェックポイント再起動データを記録するよう SAS に指示します。

- STEPCHKPTLIB システムオプションは、ユーザー指定のチェックポイント再起動ライブラリを識別します。
- STEPRESTART システムオプションでは、再起動モードを有効にできます。これにより、チェックポイント再起動ライブラリで示されている DATA ステップまたは PROC ステップで実行が再開されます。

ラベリングされたコードセクションでチェックポイントモードと再起動モードを有効にするには、SAS でバッチプログラムを起動する際にこれらのシステムオプションを使用します。

- LABELCHKPT システムオプションでは、ラベリングされたコードセクションのチェックポイントモードを有効にできます。これにより、チェックポイント再起動データを記録するように SAS に指示します。
- LABELCHKPTLIB システムオプションは、ユーザー指定のチェックポイント再起動ライブラリを識別します。
- LABELRESTART システムオプションでは、再起動モードを使用できます。これにより、チェックポイント再起動ライブラリで示されているラベリングされたコードセクションで実行が再開されます。

チェックポイント再起動ライブラリとして Work ライブラリを使用する場合、CHKPTCLEAN システムオプションを使用すると、バッチプログラムの実行に成功した後で Work ライブラリのファイルを消去できます。

詳細については、次のシステムオプション(*SAS System Options: Reference*)を参照してください。

- “STEPCHKPT System Option” (*SAS System Options: Reference*)
- “STEPCHKPTLIB= System Option” (*SAS System Options: Reference*)
- “STEPRESTART System Option” (*SAS System Options: Reference*)
- “LABELCHKPT System Option” (*SAS System Options: Reference*)
- “LABELCHKPTLIB= System Option” (*SAS System Options: Reference*)
- “LABELRESTART System Option” (*SAS System Options: Reference*)
- “CHKPTCLEAN System Option” (*SAS System Options: Reference*)

チェックポイントモードと再起動モードの使用の必要条件

チェックポイントモードと再起動モードが正常に動作できるようにするには、バッチプログラムの DATA ステップと PROC ステップまたはラベリングされたコードセクションの数と順番は、SAS を呼び出すたびに変更しないでください。SAS の起動時に ERRORABEND と ERRORCHECK システムオプションを指定することで、SAS は、有効なチェックポイント再起動データを保持するために、ほとんどのエラー状態で終了します。

チェックポイント再起動ライブラリには、ユーザー指定ライブラリを使用できます。ライブラリが指定されていない場合、チェックポイント再起動データは Work ライブラリに保存されます。チェックポイント再起動データの保存先がユーザー指定ライブラリか Work ライブラリかに関係なく、必ず NOWORKTERM システムオプションと NOWORKINIT システムオプションで SAS を起動します。SAS は、Work ライブラリの名前を SAS ログに書き出します。

動作環境の情報

UNIX および z/OS 動作環境では、STEPCHKPT オプションまたは LABELCHKPT オプションを使用する場合、チェックポイント再起動ライブラリを常に割り当てることを検討してください。CLEANWORK ユーティリティが定期的に実行されるように設定されている場合、Work ライブラリのデータが失われる可能性

があります。z/OS では、バッチセッションで Work ライブラリを再利用することは現実的ではない可能性があります。

ラベリングされたコードセクションは一意になる必要があります。重複ラベルのラベルで再起動モードになると、SAS は最初のラベルで起動します。重複ラベル間のコードが不要に再実行される可能性があります。

チェックポイントモードと再起動モードの設定と実行

チェックポイントモードと再起動モードを設定するには、バッチプログラムに次の変更を加えます。

- バッチプログラムをサブミットするたびに実行する DATA ステップと PROC ステップの前に、CHECKPOINT EXECUTE_ALWAYS ステートメントを追加します。
- チェックポイント再起動ライブラリがユーザー定義されている場合、チェックポイント再起動ライブラリ参照名をバッチプログラムで最初のステートメントとして定義する LIBNAME ステートメントを追加する必要があります。Work ライブラリをチェックポイントライブラリとして使用する場合、LIBNAME ステートメントは不要です。

バッチプログラムが変更されると、該当するシステムオプションでプログラムを起動します。

- Work ライブラリに保存されたチェックポイント再起動データの場合、これらのシステムオプションを指定するバッチ SAS セッションを開始します。
  - 使用している動作環境で必要な場合、SYSIN にバッチプログラムを指定します。
  - STEPCHKPT または LABELCHKPT はチェックポイントモードを有効にします。
  - NOWORKTERM は、SAS の終了時に Work ライブラリを保存します。
  - NOWORKINIT は、SAS の起動時に Work ライブラリを初期化しません。
  - ERRORCHECK STRICT は、LIBNAME、FILENAME、%INCLUDE、LOCK の各ステートメントでエラーが発生した場合に、SAS を構文チェックモードに切り替えます。
  - ERRORABEND は、エラー時に SAS を終了するかどうかを指定します。
  - CHKPTCLEAN は、バッチプログラムが正常に実行された場合、Work ライブラリのファイルを削除するか、Work ライブラリを削除するかを指定します。

  Windows 動作環境では、次の SAS コマンドによって、Work ライブラリをチェックポイント再起動ライブラリとして使用し、チェックポイントモードでバッチプログラムを開始します。

```
sas -sysin 'c:\mysas\myprogram.sas'-stepchkpt -noworkterm -noworkinit
    -errorcheck strict -errorabend -chkptclean
```

- ユーザー指定ライブラリに保存されたチェックポイント再起動データの場合、これらのシステムオプションを含むバッチ SAS セッションを開始します。
  - 使用している動作環境で必要な場合、SYSIN にバッチプログラムを指定します。
  - STEPCHKPT または LABELCHKPT はチェックポイントモードを有効にします。
  - STEPCHKPTLIB または LABELCHKPTLIB は、SAS がチェックポイント再起動データを保存するライブラリのライブラリ参照名を指定します。
  - NOWORKTERM は、SAS の終了時に Work ライブラリを保存します。

- NOWORKINIT は、SAS の起動時に Work ライブラリを初期化しません。
- ERRORCHECK STRICT は、LIBNAME、FILENAME、%INCLUDE、LOCK の各ステートメントでエラーが発生した場合に、SAS を構文チェックモードに切り替えます。
- ERRORABEND は、エラー時に SAS を終了するかどうかを指定します。

Windows 動作環境では、次の SAS コマンドによって、ユーザー指定のチェックポイント再起動ライブラリを使用し、チェックポイントモードでバッチプログラムを開始します。

```
sas -sysin 'c:\mysas\myprogram.sas' -labelchkpt -labelchkptlib mylibref
    -noworkterm -noworkinit -errorcheck strict -errorabbend
```

この場合、MyProgram.sas の最初のステートメントは、**MyLibref** ライブラリ参照名を定義する LIBNAME ステートメントです。

バッチプログラムの再起動

Work ライブラリに保存されたチェックポイント再起動データを使用してバッチ SAS セッションを再度サブミットするには、SAS の起動時にこれらのシステムオプションを含めます。

- 使用している動作環境で必要な場合、SYSIN にバッチプログラムを指定します。
- STEPCHKPT または LABELCHKPT は、チェックポイントモードを続行します。
- STEPRESTART または LABELRESTART は、再起動モードを有効にし、チェックポイント再起動データを使用するよう SAS に指示します。
- NOWORKINIT は、前の SAS セッションから Work ライブラリを使用して SAS を起動します。
- NOWORKTERM は、SAS の終了時に Work ライブラリを保存します。
- ERRORCHECK STRICT は、LIBNAME、FILENAME、%INCLUDE、LOCK の各ステートメントでエラーが発生した場合に、SAS を構文チェックモードに切り替えます。
- ERRORABEND は、エラー時に SAS を終了するかどうかを指定します。
- CHKPTCLEAN は、バッチプログラムを正常に実行した場合、Work ライブラリのファイルを消去するかどうかを指定します。

Windows 動作環境では、次の SAS コマンドは、チェックポイント再起動データが Work ライブラリに保存されたバッチプログラムを再度サブミットします。

```
sas -sysin 'c:\mysas\mysasprogram.sas' -stepchkpt -steprestart -noworkinit
    -noworkterm  -errorcheck strict -errorabend -chkptclean
```

NOWORKTERM システムオプションを指定し、STEPCHKPT または LABELCHKPT のどちらかのシステムオプションを指定すると、バッチプログラムの再起動時にチェックポイントモードは引き続き有効です。

ユーザー指定ライブラリに保存されたチェックポイント再起動データを使用してバッチ SAS セッションを再度サブミットするには、SAS の起動時にこれらのシステムオプションを含めます。

- 使用している動作環境で必要な場合、SYSIN にバッチプログラムを指定します。
- STEPCHKPT または LABELCHKPT は、チェックポイントモードを続行します。
- STEPRESTART または LABELRESTART は、再起動モードを有効にし、チェックポイント再起動データを使用するよう SAS に指示します。

- STEPCHKPTLIB または LABELCHKPTLIB は、チェックポイント再起動ライブラリのライブラリ参照名を指定します。
- NOWORKTERM は、SAS の終了時に Work ライブラリを保存します。
- NOWORKINIT は、SAS の起動時に Work ライブラリを初期化しません。
- ERRORCHECK STRICT は、LIBNAME、FILENAME、%INCLUDE、LOCK の各ステートメントでエラーが発生した場合に、SAS を構文チェックモードに切り替えます。
- ERRORABEND は、エラー時に SAS を終了するかどうかを指定します。

Windows 動作環境では、次の SAS コマンドは、チェックポイント再起動データがユーザー指定ライブラリに保存されたバッチプログラムを再度サブミットします。

```
sas -sysin 'c:\mysas\mysasprogram.sas' -labelchkpt -labelrestart -labelchklib
    -noworkterm -noworkinit mylibref -errorcheck strict -errorabend
```

システムオプションを用いたエラー処理の管理

次のシステムオプションを使用することで、プログラムのエラー処理を制御したり、エラーを解決したりできます。

BYERR
SORT プロシジャが _NULL_ データセットを処理しようとしたときに SAS がエラーを生成するかどうかを指定します。

CHKPTCLEAN
チェックポイントモードまたはリセットモードでは、バッチプログラムが正常に実行された場合に Work ディレクトリのファイルを消去するかどうかを指定します。

DKRICOND=
DROP=、KEEP=、RENAME=データセットオプションの処理中、入力データセットに変数がないときにレポートするためのエラー検出のレベルを指定します。

DKROCOND=
DROP=、KEEP=、RENAME=データセットオプションの処理中、出力データセットに変数がないときにレポートするためのエラー検出のレベルを指定します。

DSNFERR
SAS データセットが見つからない場合、SAS がエラーメッセージを発行するかどうかを指定します。

ERRORABEND
SAS が終了してエラーに対応するかどうかを指定します。

ERRORCHECK=
LIBNAME、FILENAME、%INCLUDE、LOCK の各ステートメントでエラーが見つかった場合に、SAS を構文チェックモードに切り替えるかどうかを指定します。

ERRORS=
SAS が完全なエラーメッセージを発行するオブザベーションの最大数を指定します。

FMTERR
変数の形式が見つからない場合、エラーを生成するか、処理を続行するかを指定します。

INVALIDDATA=
無効な数値データが検出されたとき、変数に割り当てる値を指定します。

LABELCHKPT
ラベリングされたコードセクションを含むバッチプログラムに対して、SAS チェックポイント再起動データを記録するかどうかを指定します。

LABELCHKPTLIB
ラベリングされたコードセクションのチェックポイント再起動データを保存するライブラリのライブラリ参照名を指定します。

LABELRESTART
ラベリングされたコードセクションのチェックポイント再起動データを使用してバッチプログラムを実行するかどうかを指定します。

MERROR
マクロ名が適切なマクロキーワードと一致しないときに、警告メッセージを表示するかどうかを指定します。

QUOTELENMAX
引用符付きの文字列が許容最大長を超えた場合、SAS が警告メッセージを SAS ログに書き出すかどうかを指定します。

SERROR
マクロ変数参照がマクロ変数と一致しないときに、警告メッセージを表示するかどうかを指定します。

STEPCHKPT
チェックポイント再起動データをバッチプログラム用に記録するかどうかを指定します。

STEPCHKPTLIB=
チェックポイント再起動データを保存するライブラリのライブラリ参照名を指定します。

STEPRESTART
チェックポイント再起動データを使用してバッチプログラムを実行するかどうかを指定します。

VARINITCHK=
変数が初期化されていない場合に DATA ステップの処理を停止するか続行するかを指定します。また、SAS ログに書き出されるメッセージの種類も指定できます。

VNFERR
BY 変数があるデータセットに存在し、別のデータセットには存在していない場合、SAS がエラーまたは警告を発行するかどうかを指定します。このエラーまたは警告は、SET、MERGE、UPDATE、または MODIFY ステートメントを処理する場合にのみ発行されます。

SAS システムオプションの詳細については、*SAS System Options: Reference* を参照してください。

リターンコードの使用

一部の動作環境では、リターンコードがシステムに渡されます。リターンコードにアクセスする方法は、動作環境によって異なります。

動作環境の情報
リターンコードの詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

その他のエラーチェックオプション

プログラミングエラーを判別するために、次の方法を使用できます。

- 自動変数_IORC_(およびそれに関連付けられている IORCMSG 関数)を利用します。この変数は、MODIFY ステートメントを使用するとき、または SET ステートメント内で KEY=データセットオプションを使用するときに作成されます。
- ERRORS=システムオプションを利用すると、同一のエラーが SAS ログに書き出される回数を制限できます。
- SYSRC 関数および SYSMSG 関数を使用すると、データセットや外部ファイルにアクセスするための関数にエラーが発生した場合に情報を返すことができます。
- SYSRC 自動マクロ変数を使用すると、リターンコードを受け取ることができます。
- SYSERR 自動マクロ変数を使用すると、メモリ不足やコンポーネントシステムの障害など、主なシステムエラーを検出できます。
- ログコントロールオプション:

  MSGLEVEL=
  SAS ログに書き出すメッセージの詳細レベルを制御します。

  PRINTMSGLIST
  SAS ログに書き出すメッセージリストの詳細度を制御します。

  SOURCE
  ソーステートメントを SAS ログに書き出すかどうかを制御します。

  SOURCE2
  %INCLUDE でインクルードされたソーステートメントを SAS ログに書き出すかどうかを制御します。

DATA ステップにおける論理エラーのデバッグ

DATA ステップの論理エラーをデバッグする場合、DATA ステップデバッガを使用できます。この機能を使用すると、DATA ステップのステートメントを 1 行ずつ実行し、一時停止して結果の変数値をウィンドウに表示できます。表示された結果について検討することで、論理エラーが発生している箇所を突き止めることができます。デバッガは対話型なので、単一のデバッグセッションで、コマンドの発行や結果の検討などのプロセスを必要な回数だけ繰り返すことができます。デバッガを起動するには、DATA ステートメントに DEBUG オプションを追加して SAS プログラムを実行します。DATA ステップデバッガの使用方法の詳細については、*SAS データセットオプション: リファレンス*を参照してください。

9 章

SAS 出力

SAS 出力の定義

SAS 出力は、SAS プログラムを実行した結果です。ほとんどの SAS プロシジャと一部の DATA ステップアプリケーションは、実行結果として SAS 出力を作成します。SAS プログラムは、次の種類の出力の一部または全部を生成できます。

プログラム結果

SAS プロシジャおよび DATA ステップアプリケーションを実行したときに得られるプログラムの出力結果です。プログラムの実行結果は、ファイルに送信することも、レポートとして出力することもできます。さまざまなオプション、形式、ステートメント、コマンドがあり、出力をカスタマイズできます。ODS (Output Delivery System)では、出力を格納する場所と方法、出力の構造、出力のスタイルを制御するために出力先、テーブル定義、スタイル定義を指定できます。詳細については、*SAS Output Delivery System: User's Guide* を参照してください。

SAS プログラムを実行して得られる出力には次のような種類があります。

- SAS データセット
- Web で表示する HTML ファイル
- 単純なリスト用レポート
- Microsoft Word での表示に適した RTF 出力

- モバイルデバイスでの表示に適した SVG 出力
- 複数の出力先を一度に指定するための ODS ドキュメント
- PostScript および PDF などの高解像度プリンタ用にフォーマットされた出力
- (HTML に加え)さまざまなマークアップ言語でフォーマットされた出力

SAS ログ出力

SAS ログ

SAS ログには、SAS セッションの説明と、実行したソースコードのステートメントが出力されます。SAS システムオプションの設定、SAS System の実行方法(実行モード)、実行したプログラムステートメントによって多少の違いはありますが、SAS ログには次のような情報が表示されます。

- プログラムステートメント
- 作成されたデータセットの名前
- プログラム実行中に生成された説明(NOTE)、警告(WARNING)、エラー(ERROR)メッセージ
- 各データセットに含まれる変数とオブザベーションの数
- 各ステップの処理時間

SAS ステートメントを使用すると、SAS ログに特定の情報(変数値や文字列など)を出力することもできます。詳細については、“すべてのモードでのログへの書き出し” (159 ページ)を参照してください。

SAS ログは、ユーティリティ機能を実行する一部の SAS プロシジャ(DATASETS プロシジャや OPTIONS プロシジャなど)でも使用されます。*Base SAS Procedures Guide* を参照してください。

SAS ログは、プログラムによる処理の記録でもあり、デバッグ時には欠かせない機能です。ただし、SAS プログラムのデバッグで SAS ログを効果的に利用するには、特定のシステムオプションを有効にする必要があります。“ログのカスタマイズ” (159 ページ)には、使用できる SAS システムオプションについての説明があります。

SAS コンソールログ

情報、警告、エラーメッセージを記録するための定期 SAS ログがアクティブでない場合に作成されます。SAS ログがアクティブの場合、SAS コンソールログは、致命的なシステム初期化エラーまたは遅い終了メッセージの場合にのみ使用されます。

注: SAS コンソールログの出力先の詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

SAS ログ機能の出力

SAS ログ機能を使用して作成するログメッセージを含みます。SAS プログラム内でログ機能メッセージを作成できます。または、SAS サーバーメッセージをログに記録するために SAS によって作成されることもあります。ログ機能のログメッセージは、SAS プログラムの認証、管理、パフォーマンス、カスタマイズされたメッセージなどのメッセージカテゴリに基づいています。SAS プログラムでは、ログ機能関数、自動呼び出しマクロ、DATA ステップコンポーネントオブジェクトを使用してログ機能環境を作成します。

ログ機能環境は、ロガー、appender、ログイベントで構成されます。ロガーは、メッセージカテゴリを定義し、1 つ以上の appender を参照し、ロガーのメッセージレベルしきい値を指定します。メッセージレベルしきい値には、trace、debug、info、warn、error、fatal (低い順から高い順)のいずれかを使用できます。appender は、ログメッセージとメッセージの形式を書き込む物理ロケーションを定義します。ログイ

ベントは、ログメッセージ、メッセージしきい値、ロガーで構成されます。ログイベントは、SAS サーバーと SAS プログラムで開始されます。

SAS はログ機能のログイベントを処理する際に、ログイベントのメッセージレベルを、ログイベントで指定されたロガーのメッセージしきい値と比較します。ログイベントメッセージしきい値がロガーのメッセージしきい値以上の場合、ロガー定義で参照されている appender で指定されたロケーションにメッセージが書き込まれます。ログイベントがロガーによって受け付けられない場合、メッセージは破棄されます。

appender は、マクロプログラムまたは DATA ステップの間に定義されます。ロガーは、SAS セッションの間に定義されます。

詳細については、*SAS Logging: Configuration and Programming Reference* を参照してください。

SAS 出力先の指定とカスタマイズ

デフォルト出力先

定義

SAS での*出力先*とは、ODS (Output Delivery System)が特定の種類の出力を生成するために使用する宛先のことです。つまり、ODS が出力をどのように転送するかを示すものです。たとえば、ODS は出力を HTML としてブラウザに転送することも、ファイルに転送することも、単純なリストレポートとして端末やディスプレイに転送することもできます。出力先は、次の要素で決まります。

- 動作環境
- SAS の実行モード
- SAS のバージョン

デフォルトの出力先

SAS 9.3 以降のバージョンでは、Microsoft Windows および UNIX 動作環境のウィンドウモードで SAS を実行している場合、出力はデフォルトで HTML 出力先(HTML がデフォルト出力先)に送信されます。また、SAS 9.3 以降のバージョンを UNIX および Windows のウィンドウ環境で実行している場合、ODS Graphics がデフォルトで有効になります。

ただし、SAS をバッチモードで実行する場合、SAS 9.4 以前のバージョンでは LISTING(リスト)がデフォルトの出力先であり、ODS Graphics はデフォルトで無効になっています。SAS バージョンと動作モードに基づく出力先の比較については、表 9.1 (148 ページ) を参照してください。デフォルトは、レジストリまたは構成ファイルの設定によって異なることがあります。

次の表に、SAS のバージョンに基づく動作方法ごとのデフォルト出力先を示します。

表 9.1 デフォルト出力先の比較

SAS バージョン	SAS 実行モード	ビューア	ODS 出力先
SAS 9.3 以降	ウィンドウモード	**SAS 結果ビューア**またはブラウザウィンドウ	HTML
	対話型ラインモード	端末ディスプレイ	LISTING
	非対話型モード	動作環境によって異なります	
	バッチモード	動作環境によって異なります	
SAS 9.2	ウィンドウモード	**SAS アウトプットウィンドウ**	LISTING
	対話型ラインモード	端末ディスプレイ	LISTING
	非対話型モード	動作環境によって異なります	
	バッチモード	動作環境によって異なります	

動作環境の情報

SAS 出力のデフォルトの出力先は、動作環境によって異なります。構成ファイルとレジストリ設定も、出力の送信先に影響します。デフォルト出力先の詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

- UNIX: “The Default Routings for the SAS Log and Procedure Output in UNIX Environments” (*SAS Companion for UNIX Environments*)
- z/OS: “Destinations of SAS Output Files” (*SAS Companion for z/OS*)
- Windows: “Managing SAS Output under Windows” (*SAS Companion for Windows*)

新しいデフォルトと ODS 出力先の詳細については、*SAS Output Delivery System: User's Guide* を参照してください。

出力先の変更

概要

SAS では、ログ、プロシジャ、DATA ステップ出力の送信先をさまざまな方法で制御できます。実際に使用する方法は、オペレーティングシステムと SAS を実行しているモードで決まります。出力先を変更するための一般的な方法のリストについては、表 9.2 (149 ページ) を参照してください。出力は、PC や端末、プリンタ、外部ファイルなどに直接送ることができます。出力先を指定するには、SAS プロシジャ、システムオプション、コマンド、ステートメント、グローバル ODS ステートメントを使用します。

ODS を用いた出力先の変更

ODS が SAS 7 に導入される前、ほとんどのプロシジャは、従来型のラインプリンタ向けの出力を生成していました。また、その出力は、単純なリストレポートとしてリスティングウィンドウに直接送られていました。出力先を変更する場合、PROC PRINTTO および FILENAME ステートメントなどを使用していました。これらの方法は有用ですが、

ODS ではさらに多数の出力制御オプションが使用可能です。ODS 出力先ステートメントを使用すると、さまざまな形式と送信先を指定できます。

次のリストでは、出力先の指定に使用される一般的な ODS ステートメントの一部とその他の SAS 言語要素を示します。

表 9.2 *出力先の変更*

使用方法	出力結果
PRINTTO プロシジャ	DATA ステップ、SAS ログ、プロシジャの出力先を、システムのデフォルトの出力先から、指定した出力先へと変更します。PRINTTO プロシジャは ODS 出力先以外の出力先を定義します。
FILENAME ステートメント	外部ファイルまたは出力デバイスにファイル参照名を割り当てます。このとき、ファイルとデバイスの属性を指定できます。
FILE コマンド:Windows	SAS ウィンドウ環境からコマンドが発行されたときに、ログウィンドウまたはアウトプットウィンドウの内容を、指定されたファイルに保存します。
ODS LISTING ステートメント	リスト出力先の開閉、管理を行います。
ODS OUTPUT ステートメント	出力オブジェクトから SAS データセットを生成します。また、OUTPUT 出力先に出力するオブジェクトと出力しないオブジェクトのリストを管理します。
ODS DOCUMENT ステートメント	さまざまな方法でデータを再構成、ナビゲート、再生できる ODS ドキュメントを生成します。また、解析を再実行したりデータベースクエリを繰り返したりしなくても、複数の出力先を指定できます。
ODS HTML ステートメント	HTML 出力先の開閉、管理を行います。スタイルシートが埋め込まれている HTML 4.0 出力が生成されます。
ODS MARKUP ステートメント	MARKUP 出力先の開閉、管理を行います。さまざまなマークアップ言語のいずれかを使用してフォーマットされた SAS 出力が生成されます。
ODS PRINTER ステートメント	PDF 出力先の開閉、管理を行います。Adobe Acrobat などのアプリケーションで読み込まれる PDF 出力が生成されます。
ODS RTF ステートメント	RTF 出力先の開閉、管理を行います。Microsoft Word 2002 で使用されるリッチテキスト形式で出力が書き込まれます。
SAS システムオプション	SAS プログラム全体にわたって、SAS ログおよび SAS 出力の出力先を再定義します。SAS システムオプションは、SAS System の起動時に指定します。出力先の指定に使用できるシステムオプションは、ALTLOG=、ALTPRINT=、LOG=、PRINT= システムオプションです。

グローバル ODS ステートメントの概念については、次の資料を参照してください。

- “Destination Category Table” (*SAS Output Delivery System: User's Guide*).
- “Types of ODS Statements” (*SAS Output Delivery System: User's Guide*).

動作環境の情報

z/OS と UNIX 動作環境のデフォルト出力場所の変更については、次の資料を参照してください。

- z/OS:“Directing SAS Log and SAS Procedure Output” (*SAS Companion for z/OS*). および “Changing the Default Destination” (*SAS Companion for z/OS*)
- UNIX: “Changing the Default Routings in UNIX Environments” (*SAS Companion for UNIX Environments*)

出力のカスタマイズ

出力の説明

SAS には、出力をカスタマイズするためのさまざまなステートメントとシステムオプションがあります。わかりやすいタイトル、フットノート、ラベルを追加して出力をカスタマイズし、情報をページにどのようにレイアウトするかを制御することができます。

次に、使用可能なステートメントおよび SAS システムオプションの一部を示します。

表 9.3 *出力をわかりやすくするための方法*

SAS 言語要素	機能
CENTER | NOCENTER システムオプション	出力を中央揃えにするかどうかを制御します。デフォルトでは CENTER システムオプションの指定により、タイトルとプロシジャ出力は、ページとパーソナルコンピュータディスプレイに中央揃えで表示されます。
DATE | NODATE システムオプション	日時値の出力を制御します。DATE システムオプションを指定すると、各ページの先頭に SAS ジョブの開始日時が出力されます。SAS System を対話型モードで実行している場合は、ジョブの開始日時は SAS セッションの開始日時になります。
FOOTNOTE ステートメント	出力される各ページの下部にフットノートを出力します。**FOOTNOTES** ウィンドウを使用しても同様の設定ができます。
FORMCHAR=システムオプション	CALENDAR、FREQ、REPORT、TABULATE などのプロシジャで使用されるデフォルトの罫線文字を指定します。
FORMDLIM=システムオプション	CALENDAR、FREQ、REPORT、TABULATE などのプロシジャで使用されるデフォルトの罫線文字を指定します。

SAS 言語要素	機能
LABEL ステートメント	変数にラベルを割り当てます。ほとんどのプロシジャ出力では、変数名ではなくラベルが出力されます。 LABEL ステートメントを使用すると、特定の SAS プロシジャで使用されるラベルを指定することもできます。特定のプロシジャで LABEL ステートメントを使用する方法については、*Base SAS Procedures Guide* を参照してください。
LINESIZE=システムオプションと PAGESIZE=システムオプション	出力の 1 ページの行数(ページサイズ)と 1 行の文字数(行サイズ)について、デフォルト設定を変更します。デフォルト設定は、SAS System の実行方法や特定の SAS システムオプションの設定によって異なります。行数と文字数は、OPTIONS ステートメントまたは **OPTIONS** ウィンドウで指定します。DATA ステップ出力の行数と文字数は、FILE ステートメントで指定することもできます。 LINESIZE=システムオプションと PAGESIZE=システムオプションに指定した値によっては、一部の SAS プロシジャによる SAS 出力の外観が大幅に変わることがあります。
NUMBER | NONUMBER システムオプションと PAGENO=システムオプション	ページ番号の出力を制御します。NUMBER システムオプションは、印刷される出力の各ページ先頭のタイトル行に、ページ番号を印刷するかどうかを制御します。また、PAGENO=システムオプションを使用すると、SAS 出力の次のページに割り当てられる開始ページ番号を指定できます。
グローバル ODS ステートメント	出力にスタイルを適用したり、スタイルまたはテーブル定義を使用したりできます。
TITLE ステートメント	出力される各ページの上部にタイトルを出力します。デフォルトでは、次のタイトルが出力されます。**`The SAS System`** TITLE ステートメントまたは **TITLES** ウィンドウを使用すると、デフォルトのタイトルを置き換えたり、SAS プログラム用の説明タイトルを指定したりできます。TITLE ステートメントを表示しない場合は、タイトルが null のステートメント(**`title;`**)を使用します。

ODS を用いた出力のスタイルと構造のカスタマイズ

ODS を使用すると、出力先を制御するだけでなく、出力の構造やスタイルをカスタマイズできます。ODS がテーブルとスタイルのテンプレート(定義)を使用してプロシジャや DATA ステップの結果を表示するため、カスタマイズされた*テーブル*および*スタイルテンプレート*を作成することによってその結果を制御できます。.また、最初から定義を作成するつもりがない場合は、既存のスタイルとテーブル定義を変更することもできます。

ODS (Output Delivery System)の詳細については、*SAS Output Delivery System: User's Guide* を参照してください。

出力の値の再フォーマット

特定の SAS ステートメント、プロシジャ、オプションでは、出力形式を指定して値を出力できます。ウィンドウ環境では、**プロパティ**ウィンドウを使用して値の表示方法を制御できます。出力形式を適用または変更するには、FORMAT ステートメントと ATTRIB ステートメントを使用するか、SAS ウィンドウ環境では**プロパティ**ウィンドウを使用できます。

FORMAT プロシジャを使用すると、独自の出力形式および入力形式を作成して、値の表示形式を柔軟に変更することができます。FORMAT プロシジャの詳細については、“FORMAT” (*Base SAS Procedures Guide*) を参照してください。それ以外のすべての SAS システムオプションについては、*SAS System Options: Reference* を参照してください。

欠損値の印刷

SAS リストでは、通常の数値欠損値は 1 個のピリオド(.)で表されます。文字欠損値は 1 個のブランクで表されます。数値変数用に特殊欠損値を指定した場合は、アルファベットまたはアンダースコア(_)が出力されます。文字変数の場合は、変数の長さと同じ数のブランクが出力されます。

MISSING=システムオプションを使用すると、通常の数値欠損値を表すものとして、ピリオド(.)ではなく、指定した文字を出力させることができます。詳細については、“MISSING= System Option” (*SAS System Options: Reference*)を参照してください。

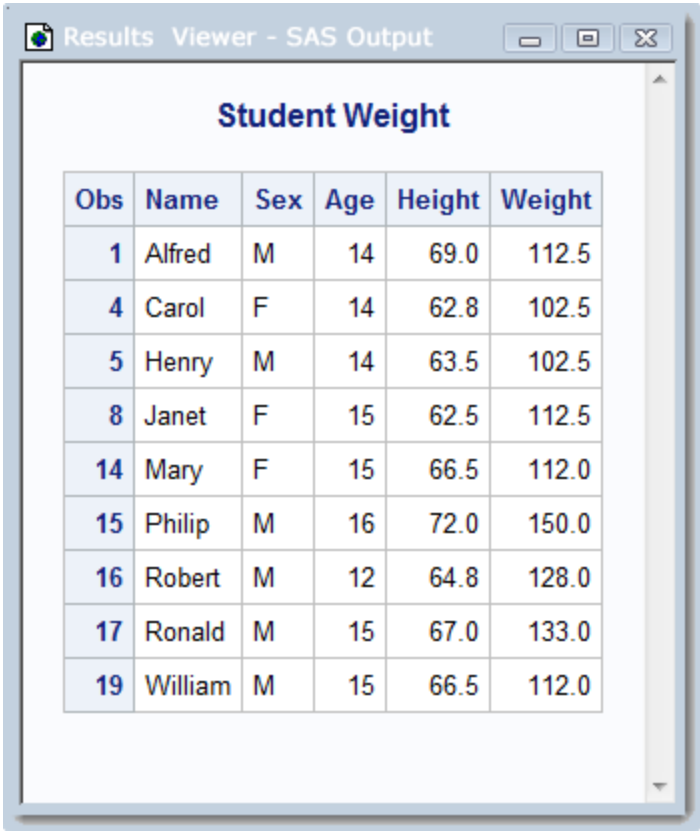
サンプル SAS 出力

SAS ウィンドウ環境におけるデフォルト HTML 出力

SAS 9.3 から、Windows および UNIX 動作環境で SAS ウィンドウ環境を実行している場合のデフォルト出力先は HTML になりました。デフォルト出力は、**SAS Results Viewer Window** に表示されます。

```
title 'Student Weight';
   proc print data=sashelp.class;
      where weight>100;
   run;
quit;
```

アウトプット 9.1 *ウィンドウ環境のデフォルト HTML 出力*



Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
8	Janet	F	15	62.5	112.5
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
19	William	M	15	66.5	112.0

注: SAS の起動時に、HTML の出力先を閉じていない限り、出力はデフォルトで WORK ディレクトリに送信されます。HTML の出力先を開き、同じ SAS セッションでもう一度開いた場合、すべての出力は WORK ディレクトリではなく現在のディレクトリに送信されます。出力を表示するために **ODS HTML CLOSE;**を指定する必要はありません。.

SAS ウィンドウ環境における従来の SAS LISTING 出力

ウィンドウモードで SAS を実行しており、出力を LISTING 出力先に送信する場合、ODS ステートメントを使用することで出力先を変更できます。さらに恒久的な解決策が必要な場合、SAS を実行するたびに設定を変更できます。出力結果はデフォルトで LISTING 出力先に送信されます。これらの設定の変更方法については、“Default HTML Output” (*SAS Output Delivery System: User's Guide*)を参照してください。

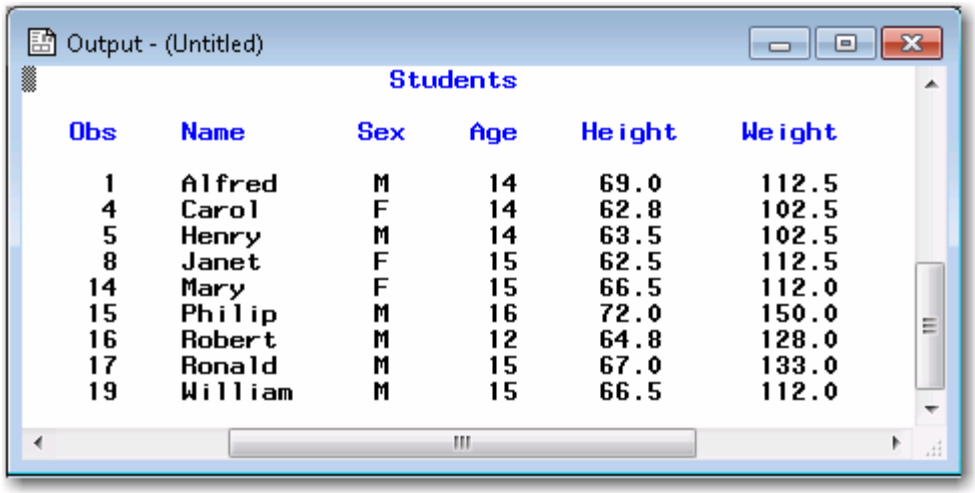
この例では、**ODS LISTING** と **ODS HTML CLOSE** ステートメントを指定することにより、出力先が HTML から LISTING に変更されます。出力先を LISTING に変更すると、出力は **SAS アウトプットウィンドウ**内にリストレポートとして自動的に表示されます。

```
ods html close;
ods listing;
options nodate;

title 'Students';
     proc print data=sashelp.class;
      where weight>100;
```

```
      run;
quit;
ods html;
ods listing close;
```

アウトプット 9.2 *ウィンドウ環境でのリスト出力*



SAS プロシジャの出力の例については、*Base SAS Procedures Guide* のプロシジャの説明を参照してください。DATA ステップ出力の説明と例については、"FILE Statement" (*SAS Statements: Reference*) および"PUT Statement" (*SAS Statements: Reference*)を参照してください。

SAS ログ

ログの構造

SAS ログは、SAS セッションまたは SAS プログラムで実行した結果をすべて記録します。オリジナルのプログラムのステートメントは、行番号で識別されます。SAS メッセージが、SAS ステートメントの行間に挿入されます。このメッセージは、NOTE、INFO、WARNING、ERROR というキーワードか、エラー番号で始まります。プログラムステートメントの問題のある箇所を調べるには、SAS ログの中に記録された行番号で参照できます。

たとえば、次の出力では、数字 1 は、OPTIONS ステートメントの左側に出力されます。この数字は、そのステートメントがプログラムの先頭の行であることを示します。対話型モードでは、行番号はセッションが終わるまで連番で振られます。プログラムを再サブミット(または現在の SAS セッションで別のプログラムをサブミット)した場合、プログラムの先頭行の番号は前からの続き番号になります。

動作環境の情報

SAS ログに出力される内容は、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

ログ 9.1 *サンプル SAS ログ*

```
NOTE:Copyright (c) 2002-2012 by SAS Institute Inc., Cary, NC, USA.1NOTE:SAS (r)
Proprietary Software 9.4 (TS1B0)  2 Licensed to SAS Institute Inc., Site
1.3NOTE:This session is executing on the W32_7PRO  platform.4NOTE:SAS
initialization used: real time           4.19 seconds cpu time            0.85
seconds 1    options pagesize=24 2    linesize=64 pageno=1 nodate;  5 3    data
logsample;  6 5       infile 5  !'\\myserver\my-directory-path
\sampledata.dat';  7 6       input LastName $ ID $ Gender $ Birth : date7.
score1 6  ! score2 score3 score4 score5 score6 score7 score8; 7       format
Birth mmddyy8.; 8    run; NOTE:The infile '\\myserver\my-directory-path
\sampledata.dat' is:8Filename=\\myserver\my-directory-path\sampledata.dat,
RECFM=V,LRECL=256,File Size (bytes)=296, Last Modified=08Jun2009:15:42:26,
Create Time=08Jun2009:15:42:26 NOTE:5 records were read from the infile  9 '\
\myserver\my-directory-path\sampledata.dat'.The minimum record length was 58.The
maximum record length was 59.NOTE:The data set WORK.LOGSAMPLE has 5 observations
and 12 variables.10NOTE:DATA statement used (Total process time): real
time           0.21 seconds  11 cpu time            0.03 seconds 9 10   proc
sort data=logsample;  12 11      by LastName; 12   run; NOTE:There were 5
observations read from the data set WORK.LOGSAMPLE.NOTE:The data set
WORK.LOGSAMPLE has 5 observations and 12 variables.13NOTE:PROCEDURE SORT used
(Total process time): real time           0.01 seconds cpu time            0.01
seconds 13 14   proc print data=logsample;  14 15      by LastName; 16   run;
NOTE:There were 5 observations read from the data set
WORK.LOGSAMPLE.NOTE:PROCEDURE PRINT used (Total process time): real
time           0.03 seconds cpu time            0.03 seconds
```

次のリストは、前述の SAS ログ内の数字に対応しています。

1 著作権情報

2 このプログラムの実行に使用された SAS System のバージョンリリース番号です。

3 プログラムが実行されたコンピュータ名とサイト番号です。

4 プログラムの実行に使用するプラットフォームです。

5 ページサイズ 24、行サイズ 64 を設定し、出力の日付を抑制するための OPTIONS ステートメントです。

6 プログラムを構成する SAS ステートメントです。SOURCE システムオプションが有効になっている場合に出力されます。

7 長いステートメントの改行です。後続する行の行頭には感嘆符(!)が表示されます。行番号は変わりません。

8 入力ファイルに関する情報です。生データと入力ソースデータについての、説明と警告メッセージです。この情報は、NOTES システムオプションが有効になっている場合に出力されます。

9 入力ファイルから読み込まれたレコードの数と長さです。NOTES システムオプションが有効になっている場合に出力されます。

10 プログラムによって作成された SAS データセット名と、そのデータセットに含まれるオブザベーションと変数の数です。NOTES システムオプションが有効になっている場合に出力されます。

11 STIMER オプションまたは FULLSTIMER オプションが設定された場合にレポートされるパフォーマンス統計量です。

12 データセットの並べ替えに使用されたプロシジャです。

13 並べ替えた SAS データセットに関する説明です。

14 データセットの出力に使用されたプロシジャです。

対話型モードでの SAS ログ

対話型モードでは、SAS ログは SAS の起動時に開かれます。SAS ログは、アクティブウィンドウに保存するまで名前が付けられません。指定した名前は、使用している動作環境のファイル命名規則に従う必要があります。SAS ログは、対話型モードでは自動的に保存できません。ただし、SAS 呼び出しまたは構成ファイルで ALTLOG=システムオプションが設定されている場合、SAS は SAS ログの 2 番目のコピーを作成できます。

バッチモード、ラインモード、オブジェクトサーバーモードでの SAS ログ

バッチモード、ラインモード、オブジェクトサーバーモードの SAS ログの概要

SAS の起動時に LOGCONFIGLOC=システムオプションが指定されていない場合、LOG=システムオプションと LOGPARM=システムオプションを使用して SAS ログを構成できます。これらのオプションは、バッチモード、ラインモード、オブジェクトサーバーモードのいずれかで指定できます。LOGCONFIGLOC=システムオプションが指定されている場合、SAS ログ機能によってログが実行され、LOGPARM=システムオプションは無視されます。LOG=オプションは、%S{App.Log}変換文字がログ構成ファイルで指定された場合にのみ使用できます。

以降のセクションでは、LOGPARM=システムオプションを使用して構成可能なログオプション、およびログ機能が開始していない場合にそのオプションに対してどのように SAS ログ名を付けるかについて説明します。

LOG=システムオプションは、SAS ログに名前を付けます。LOGPARM=システムオプションでは、次のタスクを実行できます。

- 既存の SAS ログを追加または置換する
- SAS ログに表示するタイミングを判定する
- 一定の条件下で新しい SAS ログを開始する

ログシステムオプションの詳細については、お使いの動作環境用のドキュメントに記載されている“LOGPARM= System Option” (*SAS System Options: Reference*) を参照してください。SAS のログ機能の詳細については、*SAS Logging: Configuration and Programming Reference* を参照してください。

SAS ログへの追加または SAS ログの置換

LOG=システムオプションで SAS ログの出力先を指定した場合、SAS ログがすでに存在するかどうかが検証されます。ログが存在する場合は、LOGPARM=システムオプションの OPEN=オプションで SAS ログへの表示方法を指定できます。

OPEN=APPEND
SAS ログの内容を既存の SAS ログに追加します。

OPEN=REPLACE
既存の SAS ログを置き換えます。

OPEN=REPLACEOLD
24 時間経過した既存の SAS ログを置き換えます。

次の SAS コマンドでは、1 日経過した既存の SAS ログを置き換えるために LOG=システムオプションと LOGPARM=システムオプションが指定されています。

```
sas -sysin "my-batch-program"  -log "c:\sas\SASlogs\mylog"
    -logparm open=replaceold
```

LOGPARM=システムオプションの ROLLOVER=オプションが特定のサイズ *n* に設定されている場合、OPEN=オプションは無視されます。

SAS ログに書き出すタイミングの指定

SAS ログの内容が生成された時点でその内容をログに書き出すこともできれば、または内容をいったんバッファし、バッファがいっぱいになった時点で書き出すこともできます。デフォルトでは、ログのバッファがいっぱいになったときにログに書き出されます。ログの内容をバッファすることで、一度に 1 行ずつではなく定期的にログファイルに書き出されるので、SAS のパフォーマンスはより効率的になります。

Windows 固有

Windows では、バッファされたログの内容が、SAS で指定した間隔で定期的に書き出されます。

LOGPARM=システムオプションの WRITE=オプションを使用すると、SAS ログの内容をいつ表示するかを設定できます。LOGPARM="WRITE=IMMEDIATE"と設定すると、ログが生成された時点で書き出されます。LOGPARM="WRITE=BUFFERED"と設定すると、バッファがいっぱいになったときに書き出されます。

SAS ログのロールオーバー

SAS ログのロールオーバーの概要:長時間実行しているサーバーやバッチジョブの場合、SAS ログは非常に大きくなる可能性があります。LOGPARM=システムオプションと LOG=システムオプションを一緒に使用すると、SAS ログを新しい SAS ログにロールオーバーできます。ログがロールオーバーされると、ログが閉じられ、新しいログが開かれます。

LOGPARM=システムオプションはログファイルがいつ開かれていつ閉じられるかを制御し、LOG=システムオプションは SAS ログファイルに名前を付けます。SAS セッションが開始したとき、ログが特定のサイズに達したとき、またはログがサイズに達しなかった場合、ログを自動的にロールオーバーできます。SAS ログ名のフォーマット用ディレクティブを使用すると、各 SAS ログを一意の識別子で命名できます。

SAS ログを命名するディレクティブの使用方法:SAS ログの場合、ディレクティブは、SAS ログを一意に命名するための処理指令です。ディレクティブを使用すると、日付、時刻、システムノード名、一意の識別子などの情報を SAS ログに追加できます。LOG=システムオプションでログ名を含めると、SAS ログ名には 1 つ以上のディレクティブを指定できます。たとえば、SAS ログ名に年、月、日を含める場合、LOG=システムオプションは次のようになります。

```
-log='c:\saslog\#Y#b#dsas.log'
```

SAS ログが 2009 年 2 月 2 日に作成された場合、ログの名前は 2009Feb02sas.log となります。

ディレクティブは、LOGPARM=システムオプションの ROLLOVER=オプションの値が AUTO または SESSION に設定されている場合にのみ無効です。ログ名にディレクティブが指定されており、ROLLOVER オプションの値が NONE または特定のサイズ *n* である場合、#b や#Y などのディレクティブ文字はログ名の一部になります。LOG=システムオプションで上記の例を使用すると、LOGPARM=システムオプションで ROLLOVER=NONE を指定した場合、SAS ログ名は#Y%b#dsas.log になります。

ディレクティブの完全なリストについては、"LOGPARM= System Option" (*SAS System Options: Reference*)を参照してください。

ディレクティブ変更時の SAS ログの自動ロールオーバー:SAS ログ名に 1 つ以上のディレクティブが含まれ、LOGPARM=システムオプションの ROLLOVER=オプションが AUTO に設定されている場合、ディレクティブ値が変更されるとログが閉じられ、新しいログが開かれます。新しい SAS ログ名には、新しいディレクティブ値が含まれます。

次の表に、月の 2 日目の 6:15 AM に SAS System を起動したときに、次の SAS コマンドで作成されるログ名の一部を示します。

```
sas -objectserver -log "london#n#d#%H.log"
-logparm
"rollover=auto"
```

ディレクティブ#n は、システムノード名をログ名に挿入します。#d は、その月の日をログ名に追加します。#H は、時間をログ名に追加します。この例では、ノード名は Thames です。この SAS セッションのログは、時間や日が変更されたときにロールオーバーされます。

***表 9.4** ロールオーバー済みのログの名前*

ロールオーバー時間	ログ名
SAS 初期化	londonThames0206.log
最初のロールオーバー	londonThames0207.log
その日の最後のログ	londonThames0223.log
深夜 12 時を過ぎてから最初のログ	londonThames0300.log

SAS セッション単位での SAS ログのロールオーバー:SAS セッションの開始時にログをロールオーバーするには、SAS の起動時に LOGPARM="ROLLOVER=SESSION" オプションを指定します。SAS は、起動時に取得したシステム情報を使用してシステム固有のディレクティブを解決します。SAS セッション中にはロールオーバーが発生せず、SAS セッションの終了時にログファイルが閉じられます。

ログサイズ単位でのログのロールオーバー:ログが特定のサイズに達したときにログをロールオーバーするには、SAS System の起動時に LOGPARM="ROLLOVER=*n*"オプションを指定します。*n* は、ログの最大サイズ(バイト)です。10K (10,240)バイトより小さくすることはできません。指定したサイズにログが達した場合、SAS はログを閉じて、ファイル名に"old"というテキストを付加(たとえば londonold.log)します。SAS は、LOG=オプションの値をログ名に使用して新しいログを開きます。その際、LOGPARM=システムオプションの OPEN=オプションステートメントは無視されます。これにより、既存のログファイルが上書きされなくなります。ログサイズに基づいてロールオーバーされるログの場合、ログ名のディレクティブは無視されます。

サーバー間で一意のログファイル名を維持するために、SAS はログファイル名に基づくロックファイルを作成します。ロックファイル名は *logname*.lck であり、ここで *logname* は LOG=オプションの値です。サーバーログ用のロックファイルが存在し、別のサーバーが同じログ名を指定している場合、その別のサーバのログおよびロックファイルの名前には番号が付けられます。番号は 2 から始まり、同じログファイル名が作成されるたびに 1 ずつ増えていきます。たとえば、ログファイル london.log のロックが存在している場合、2 番目のサーバーログは london2.log、ロックファイルは london2.lck となります。

SAS ログのロールオーバーを行わない:ログをロールオーバーしない場合は、SAS の起動時に LOGPARM= "ROLLOVER=NONE"オプションを指定します。ディレクティブは展開されず、ロールオーバーは発生しません。たとえば、LOG="March#b.log"の場合、ディレクティブ#b は展開されず、ログ名は March#b.log となります。

すべてのモードでのログへの書き出し

すべてのモードで、SAS ログに追加的な情報を書き出すには、次のステートメントを使用します。

PUT ステートメント
DATA ステップの現在の反復で、選択した行(テキスト文字列と DATA ステップ変数値)を SAS ログに出力します。LOG 出力先を持つ FILE ステートメントが PUT ステートメントの前に実行された場合、PUT ステートメント出力は、FILE ステートメントで指定した出力先に送信されます。

%PUT ステートメント
文字列やマクロ変数値を SAS ログに出力します。%PUT は SAS マクロプログラムステートメントです。DATA ステップとは無関係に、任意の場所で使用することができます。

PUTLOG ステートメント
ユーザー指定メッセージを SAS ログに出力します。PUTLOG ステートメントは、DATA ステップの中で使用します。

LIST ステートメント
処理中のデータ行の入力データレコードを SAS ログに出力します。LIST ステートメントの処理は、INPUT ステートメントで読み込まれたデータに対して有効です。SET、MERGE、MODIFY、UPDATE ステートメントで読み込まれたデータには無効です。LIST ステートメントは、DATA ステップの中で使用します。

/NESTING オプション付きの DATA ステートメント
DO-END ステートメントと SELECT-END ステートメントの各ネストレベルの開始と終了に関する注記を SAS ログに表示します。これにより、不一致の DO-END ステートメントと SELECT-END ステートメントをデバッグできます。

ERROR ステートメント
自動変数_ERROR_の値を 1 に設定します。(オプション)指定したメッセージを SAS ログに表示することもできます。ERROR ステートメントは、DATA ステップの中で使用します。

PUT、PUTLOG、LIST、DATA、ERROR ステートメントを条件付き処理と組み合わせて使用することにより、任意のデバッグ情報を SAS ログに表示することができます。

ログのカスタマイズ

ログコンテンツの変更

プログラムを変更する必要のない大規模な SAS プロダクションプログラム(アプリケーション)を定期的に実行している場合は、SAS ログの一部を出力しないように非表示の設定をすることもできます。SAS システムオプションを使用すると、SAS ステートメントやシステムメッセージを SAS ログに出力しないように非表示に設定したり、エラーメッセージの数を制限したりできます。SAS システムオプションは、途中で変更を加えない限り、SAS セッションが終了するまで有効です。プログラムがエラーなく正常に実行されることを確認するまでは、メッセージの出力を抑制しないでください。

次に、ログのコンテンツを変更するための SAS システムオプションを示します。

CPUID | NOCPUID
ハードウェア情報を SAS ログに出力するかどうかを指定します。

ECHO
SAS 初期化時に SAS ログに出力するメッセージを指定します。ECHO システムオプションは、Windows および UNIX 動作環境でのみ有効です。

ECHOAUTO | NOECHOAUTO
入力ファイル内の自動実行プログラムを SAS ログに出力するかどうかを指定します。

ERRORS=*n*
エラーメッセージを表示するオブザベーションの最大数を指定します。

FULLSTATS
完全な統計量を SAS ログに出力します。FULLSTATS システムオプションは z/OS でのみ有効です。

FULLSTIMER
システムパフォーマンス統計量の一部を SAS ログに表示します。

ISPNOTES
ISPF エラーメッセージを SAS ログに表示するかどうかを指定します。ISPNOTES システムオプションは、z/OS 動作環境でのみ有効です。

HOSTINFOLONG
SAS 起動時、SAS ログに追加の動作環境情報を書き出します。

LOGPARM “OPEN=APPEND | REPLACE | REPLACEOLD”
ログファイルがすでに存在しており、ログが開かれている場合、LOGPARM オプションは、既存のログに追加するか、既存のログを置き換えるかを指定します。REPLACEOLD オプションは、1 日を超えたログを置き換えるように指定します。

MEMRPT
各ステップでメモリの使用統計量を SAS ログに出力するかどうかを指定します。MEMRPT システムオプションは、z/OS 動作環境でのみ有効です。

MLOGIC
マクロ実行トレース情報を SAS ログに表示します。

MLOGICNEST
マクロネスト実行トレース情報を SAS ログに表示します。

MPRINT | NOMPRINT
マクロの実行によって生成された SAS ステートメントを、SAS ログに表示するどうかを制御します。

MSGLEVEL=N | I
SAS ログに表示するメッセージの詳細レベルを指定します。MSGLEVEL=システムオプションに N を設定した場合、SAS ログには説明(NOTE)、警告(WARNING)、エラー(ERROR)メッセージだけが表示されます。MSGLEVEL=システムオプションに I を設定した場合、インデックスの使用、マージ処理、HADOOP MapReduce ジョブ、並べ替えユーティリティに関する補足説明が表示されます。

NEWS=*external-file*
サイトで管理されている新しい情報を、SAS ログに表示するかどうかを指定します。

NOTES | NONOTES
説明(NOTE)を SAS ログに表示するかどうかを指定します。NONOTES システムオプションを指定しても、エラーメッセージや警告メッセージは非表示に抑制されません。

OPLIST
SAS が呼び出されたときに指定したすべてのシステムオプションの値を SAS ログに表示するかどうかを指定します。

OVP | NOOVP
SAS によって出力されたエラーメッセージを重ね打ちするかどうかを指定します。

PAGEBREAKINITIAL
SAS ログとリストファイルを新しいページで始めるかどうかを指定します。

PRINTMSGLIST | NOPRINTMSGLIST
メッセージの詳細リストを SAS ログに表示するかどうかを指定します。

RTRACE
SAS 実行中に読み込まれたリソースのリストを生成し、RTRACELOC=システムオプション用の場所が指定されていない場合に SAS ログに表示します。RTRACE システムオプションは、Windows および UNIX 動作環境でのみ有効です。

SOURCE | NOSOURCE
ソースステートメントを SAS ログに表示するかどうかを指定します。

SOURCE2 | NOSOURCE2
%INCLUDE ステートメントで指定したファイルから得られた 2 次ソースステートメントを、SAS ログに表示するかどうかを指定します。

SYMBOLGEN | NOSYMBOLGEN
マクロ変数参照の展開した結果を SAS ログに表示するかどうかを指定します。

VERBOSE
構成ファイルで指定したシステムオプションの値をバッチログに表示するか、コンピュータモニタに表示するかを指定します。

前述およびその他の SAS システムオプションの使用方法の詳細については、*SAS System Options: Reference* を参照してください。

動作環境の情報
SAS ログの出力に影響を与えるその他のオプションについては、使用している動作環境に対応する SAS ドキュメントを参照してください。

ログの表示のカスタマイズ

ログをカスタマイズするには、次の SAS ステートメントと SAS システムオプションを使用します。SAS ログを利用したレポートなどを作成するときは、カスタマイズしておくと便利です。

DATE システムオプション
SAS ログ、および SAS System で作成されるすべての出力結果について、各ページの先頭に SAS ジョブの開始日時を出力するかどうかを制御します。

DETAILS | NODETAILS
SAS ライブラリに保存されているファイルの一覧を表示するときに、追加情報を表示するかどうかを指定します。

DMSLOGSIZE=システムオプション
SAS ログウィンドウに表示する最大行数を指定します。

DTRESET | NODTRESET
SAS ログまたはリストファイルの日付と時刻を更新するかどうかを指定します。

FILE ステートメント
PUT ステートメントの実行結果を外部ファイルに書き出します。FILE ステートメントで次の 2 つのオプションを使用すると、SAS ログをレポート作成用にカスタマイズできます。

LINESIZE=*value*	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指定します。
PAGESIZE=*value*	出力の各ページに印刷される最大行数を指定します。

注: FILE ステートメントで指定した出力だけに、FILE ステートメントのオプションが適用されます。一方、SAS システムオプションの LINESIZE=と PAGESIZE=は、後続するすべての出力に適用されます。

LINESIZE=システムオプション
DATA ステップおよび SAS プロシジャから出力される SAS ログと SAS 出力について、1 行の文字数(プリンタの行の幅)を指定します。

MSGCASE
注記、警告、エラーメッセージを大文字で表示するか、小文字で表示するかを指定します。

MISSING=システムオプション
数値変数の値が欠損値の場合に出力する文字を指定します。

NUMBER システムオプション
印刷される出力の各ページの先頭にあるタイトル行に、ページ番号を付けるかどうかを制御します。

PAGE ステートメント
SAS ログの新しいページにスキップして、そこから出力を続行します。

PAGESIZE=システムオプション
SAS 出力の 1 ページに出力できる行数を指定します。

SKIP ステートメント
SAS ログ中に、指定した数だけブランク行を出力します。

STIMEFMT=システムオプション
STIMER システムオプションを設定する際に読み取り時間と CPU 処理時間を表示するための形式を指定します。STIMEFMT=システムオプションは、Windows、VMS、および UNIX 動作環境でのみ有効です。

動作環境の情報
FILE ステートメントおよび SAS システムオプションに指定できる値の範囲は、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

前述およびその他の SAS システムオプションとステートメントの使用方法の詳細については、*SAS System Options: Reference* を参照してください。

SAS ログに影響を与えるその他のシステムオプション

次のシステムオプションは、SAS ログのコンテンツと表示を除く、SAS ログ関連のオプションです。

ALTLOG=システムオプション
SAS ログのコピーの出力先を指定します。

LOG=システムオプション
SAS がバッチモードで実行されている場合の SAS ログの出力先を指定します。

LOGAPPLNAME
SAS ログで使用可能な SAS セッション名を指定します。

10 章

SAS プログラムのグループ処理(BY 処理)

BY グループ処理の定義

BY グループ処理とは、1 つ以上の共通する変数の値を基準にしてグループ化または並べ替えた SAS データセットが 1 つ以上ある場合に、そのデータセットにあるオブザベーションをグループごとに処理する方法です。BY グループ処理は、DATA ステップでも PROC ステップでも使用できます。

DATA ステップにおける BY グループ処理で最も一般的なものは、BY ステートメントを SET、MERGE、MODIFY、または UPDATE ステートメントと組み合わせて複数の SAS データセットを結合するというものです。SAS プロシジャでレポートまたはサマリを作成する場合、BY グループ処理を使用すると、1 つ以上の変数の値に従って出力の情報をグループ化できます。

BY グループ処理の参照

- BY グループ処理の詳細については、20 章, “DATA ステップでの BY グループ処理” (437 ページ)を参照してください。
- SAS プロシジャで BY グループ処理を使用する方法については、“Fundamental Concepts for Using Base SAS Procedures” (*Base SAS Procedures Guide*) および *Base SAS Procedures Guide* の個々のプロシジャを参照してください。
- BY グループ処理を使用して複数の SAS データセットからの情報を結合する方法については、21 章, “SAS データセットの加工” (453 ページ) を参照してください。BY グループ処理のさまざまな例については、*Combining and Modifying SAS Data Sets:Examples* を参照してください。
- BY ステートメントの詳細については、*SAS Statements: Reference* の Statements を参照してください。
- 他のソフトウェア製品で BY グループ処理を使用する方法については、各製品の SAS ドキュメントを参照してください。

11 章

WHERE 式の処理

WHERE 式処理の定義

WHERE 式処理
: WHERE 式は、指定した条件に基づいてオブザベーションのサブセットを選択します。これによって、指定した条件を満たすオブザベーションだけを処理することができます。たとえば、売上記録を含む SAS データセットでは、売上高が 300,000 ドルより大きく 600,000 ドルより小さいオブザベーションのサブセットを出力できます。さらに、WHERE 式の処理によって、要求に対する効率が向上する場合があります。たとえば、WHERE 式がインデックスにより最適化されている場合は、要求を実行するためにデータセット内のオブザベーションのすべてを読み込む必要はなくなります。

WHERE 式
: 処理されるオブザベーションが満たす条件を定義します。次のような単一の WHERE 式を単純式と呼びます。

```
where sales gt 600000;
```

また、次のような複数の条件を含む WHERE 式を複合式と呼びます。

```
where sales gt 600000 and salary lt 100000;
```

WHERE 式の使用場所

WHERE 式は、次のような箇所で使用できます。

- DATA ステップおよび PROC ステップの WHERE ステートメント。たとえば、次の PRINT プロシジャの WHERE ステートメントでは、2001 年以降の年のオブザベーションだけを出力します。

```
proc print data=employees;
   where startdate > '01jan2001'd;
run;
```

- WHERE=データセットオプション。次の PRINT プロシジャでは、WHERE=データセットオプションを指定しています。

```
proc print data=employees (where=(startdate > '01jan2001'd));
run;
```

- SQL プロシジャ、SCL プログラム、SAS/IML ソフトウェアでの WHERE 句。次の SQL プロシジャの WHERE 句では、7 件を超える殺人事件が発生している州だけを選択しています。

```
proc sql;
   select state from crime
   where murder > 7;
```

- SAS/FSP ソフトウェアなどのウィンドウ環境における WHERE コマンド。

```
where age > 15
```

- SAS ビュー(DATA ステップビュー、SAS/ACCESS ビュー、PROC SQL ビュー)。次の SQL プロシジャは、PROC SQL ビューを STAT という名前でデータファイル Crime から作成し、PROC SQL ビュー定義のための WHERE 式を定義しています。

```
proc sql;
   create view stat as
   select * from crime
   where murder > 7;
```

WHERE 式を指定する複数の方法を組み合わせることもできます。つまり、WHERE ステートメントを次のように使用できます。

- WHERE=データセットオプションと組み合わせる。
- WHERE=データセットオプションをウィンドウプロシジャで使用し、さらに WHERE コマンドを組み合わせる。
- WHERE 式が格納されている SAS ビューで使用する。

たとえば、データセットを結合する場合、複数の方法を組み合わせることは有用です。つまり、データセットごとに異なる条件を適用してサブセットを作成できます。ただし、複数の方法を組み合わせてサブセットを作成するには、いくつかの制限があります。たとえば、DATA ステップで、WHERE ステートメントと WHERE=データセットオプションを同じデータセットに適用するときは、データセットオプションが優先されます。詳細については、WHERE 式を指定するための方法を記載したドキュメントを参照してください。

注: デフォルトでは、追加されたオブザベーションおよび変更されたオブザベーションは、WHERE 式の評価対象外です。WHERE 式で、更新されたオブザベーションを

評価するかどうかを指定するには、WHEREUP=データセットオプションを使用します。“WHEREUP= Data Set Option” (*SAS Data Set Options: Reference*)を参照してください。

WHERE 式の構文

WHERE 式の内容

WHERE 式とは、オブザベーションを選択するための条件を定義する SAS 式の一種です。WHERE 式には、単一の変数名または単一の定数(固定値)を指定できます。また、SAS 関数、またはオブザベーションを選択する条件を定義するオペランドおよび演算子を指定できます。WHERE 式の一般的な構文は次のとおりです。

WHERE *operand* <*operator*> <*operand*>

operand
オペランドは、演算処理の対象となるものです。WHERE 式のオペランドには変数、SAS 関数、定数を指定できます。“オペランドの指定” (167 ページ)を参照してください。

operator
演算子は、比較演算、論理演算、算術演算を要求する記号です。SAS 式の演算子はすべて WHERE 式で有効です。有効な演算子には、算術演算子、比較演算子、論理演算子、最小演算子、最大演算子、連結演算子、評価の順序を制御するかっこ、接頭演算子などがあります。また、WHERE 式に特有の演算子も使用できます。有効な演算子には、BETWEEN-AND、CONTAINS、IS NULL、または IS MISSING、LIKE、=*(SOUNDS-LIKE)、SAME-AND などがあります。“演算子の指定” (170 ページ)を参照してください。

オペランドの指定

変数

変数とは SAS データセット内の列です。SAS 変数には、それぞれ名前や種類(文字または数値)などの属性があります。変数の種類によって、検索する値の指定方法は異なります。次にその例を示します。

```
where score > 50;
where date >= '01jan2001'd and time >= '9:00't;
where state = 'Texas';
```

WHERE 式で使用できる変数は、DATA ステップで作成された自動変数を除きます。たとえば、FIRST.variable、LAST.variable、_N_、割り当てステートメントで作成された変数などは使用できません。

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。SAS System では、数値 0 や欠損値は偽、それ以外の値は真として扱われます。次の例では、WHERE 式は、EMPNUM が失われておらずゼロでないすべての行と、ID が失われておらずゼロではないすべての行を返します。

```
where empnum and id;
```

文字変数の名前も単独で指定できます。この場合、文字変数の値がブランクでないオブザベーションが選択されます。たとえば、次の WHERE 式では、ブランクに等しくない値がすべて返されます。

```
where lastname;
```

SAS 関数

SAS 関数は、演算またはシステム操作の結果の値を返します。ほとんどの関数では、指定した引数が使用されますが、動作環境から引数を取得する関数もあります。SAS 関数を WHERE 式で使用するには、関数名に続けて引数をかっこで囲んで記述します。次に示す関数は、WHERE 式で有用です。

- SUBSTR 関数。部分文字列を抽出します。
- TODAY 関数。現在の日付を返します。
- PUT 関数。指定した出力形式を使用して変換した値を返します。

次の DATA ステップでは、データセット Customer のオブザベーションのうち、Name の値が **Mac** で始まり、変数 City の値が **Charleston** または **Atlanta** であるもののみを含む 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;
```

OF 構文は、一部の SAS 関数で許可されていますが、WHERE 句に指定されている関数を使用する際には使用できません。次の DATA ステップ例では、OF を RANGE と組み合わせて使用しています。

```
data abc;
x1=2;
x2=3;
x3=4;
r=range(of x1-x3);
run;
```

WHERE 句に RANGE と OF を付けて使用する場合、エラーは SAS ログに書き出されます。

ログ 11.1 *WHERE 句を OF 付きで使用した場合の出力*

```
proc print data=abc; where range(of x1-x3)=6; -- 22 76 ERROR:Syntax error while
parsing WHERE clause.ERROR 22-322:Syntax error, expecting one of the
following:!, !!, &, (, *, **, +, ',', -, /, <, <=, <>, =, >, >=, ?, AND,
BETWEEN, CONTAINS, EQ, GE, GT, LE, LIKE, LT, NE, OR, ^=, |, ||, ~=.ERROR
76-322:Syntax error, statement will be ignored. run;
```

次の表に、OF 構文を使用できる SAS 関数を示します。

表 11.1 OF 構文を使用する SAS 関数

CAT	HARMEANZ	RMS
CATS	KURTOSIS	SKEWNESS
CATT	MAX	STD
CATX	MEAN	STDERR
CSS	MIN	SUM
CV	N	USS
GEOMEAN	NMISS	VAR
GEOMEANZ	ORDINAL	
HARMEAN	RANGE	

注: SAS 関数のうち、SUBSTR 関数と TRIM 関数を WHERE 式で指定すると、利用可能なインデックスが使用されます。

SAS 機能の詳細については、*SAS Functions and CALL Routines: Reference* を参照してください。

定数

定数とは、数や、引用符で囲まれた文字列などの固定値です。つまり、定数は検索する対象の値です。また、定数は SAS データセットから取得された変数の値、または WHERE 式それ自体の内部で作成された値です。定数はリテラルとも呼ばれます。たとえば、定数には航空便名や都市の名前を指定します。また、時間、日付、日時の値も指定できます。

定数の値は、数値または文字で指定します。次に、引用符の使用に関する規則を示します。

- 値が数値の場合、引用符は不要です。

```
where price > 200;
```

- 値が文字値の場合、引用符を使用します。

```
where lastname eq 'Martin';
```

- 一重引用符(')と二重引用符(")のいずれも使用できますが、混用しないようにします。引用符で囲まれた値は、アルファベットの大文字と小文字の区別を含めて、正確に一致していることが必要です。
- 値の中に二重引用符(")が含まれる場合は、一重引用符(')を使用します。同様に、値の中に一重引用符(')が含まれる場合は、二重引用符(")を使用します。

```
where item = '6" decorative pot';
where name ? "D'Amico";
```

- SAS 日付定数は引用符で囲みます。日付値を指定する場合は、アルファベットの大文字と小文字の区別なく指定できます。一重引用符(')または二重引用符(")のどちらも使用できます。次の 2 つの式は同じ意味を持ちます。

```
where birthday = '24sep1975'd;
where birthday = '24sep1975"d;
```

演算子の指定

算術演算子

算術演算子によって、算術計算を実行できます。算術演算子の種類を次の表に示します。

表 11.2 算術演算子

記号	定義	例
*	乗算	where bonus = salary * .10;
/	除算	where f = g/h;
+	加算	where c = a+b;
-	減算	where f = g-h;
**	累乗	where y = a**2;

比較演算子

比較演算子(2 項演算子とも呼ばれる)は、変数と値の比較、または変数と別の変数の比較を行います。比較演算子は関係を示し、その関係が真であるかどうかを調べます。たとえば、次の WHERE 式は、数値変数 ZipCode の値が 78753 であるオブザベーションにのみアクセスします。

```
where zipcode eq 78753;
```

比較演算子の種類を次の表に示します。

表 11.3 比較演算子

記号	ニーモニック	定義	例
=	EQ	等しい	where empnum eq 3374;
^= or ~= or ¬= or <>	NE	等しくない	where status ne full-time;
>	GT	より大きい	where hiredate gt '01jun1982'd;
<	LT	より小さい	where empnum < 2000;
>=	GE	以上	where empnum >= 3374;
<=	LE	以下	where empnum <= 3374;
	IN	値のリストのいずれかと等しい	where state in ('NC','TX');

文字列を比較する場合、コロン修飾子(:)を使用すると、文字列の一定の先頭部分だけを比較できます。たとえば、次の WHERE 式では、等号(=)の後ろにコロン修飾子(:)を記述して、変数 LastName の値のうち、先頭の文字だけを評価します。これにより、s で始まる名前を含むオブザベーションが選択されます。

```
where lastname=: 'S';
```

SQL プロシジャでは、コロン修飾子(:)は演算子と共に使用できないため、そのかわりに LIKE 演算子を使用します。

IN 演算子

IN 演算子は比較演算子であり、値のリストのいずれかと等しい文字値または数値を検索します。値のリストはかっこで囲みます。リスト内の文字値は引用符で囲みます。また、各値はカンマ(,)またはブランクのいずれかで区切ります。

たとえば、ノースカロライナ州またはテキサス州にあるすべてのサイトを検索する場合を考えます。次のように指定できます。

```
where state = 'NC' or state = 'X';
```

ただし、このような場合は IN 演算子を使用する方がより簡単です。次のように、IN 演算子を使用して州をリストに指定します。

```
where state in ('NC','TX');
```

さらに、NOT 論理演算子を使用してリストを除外することもできます。

```
where state not in ('CA', 'TN', 'MA');
```

簡略表記を使用して、検索する連続した整数の範囲を指定できます。この範囲を指定するには、検索対象とするリスト内の値として構文 M:N を使用します。ここで、M は下限、N は上限になります。M および N は整数でなければなりません。M、N、および M と N の間にあるすべての整数が範囲に含まれます。たとえば、次の 2 つのステートメントは同じ意味を持ちます。

- **y = x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);**
- **y = x in (1:10);**

上限と下限が指定された範囲条件

範囲の指定は、1 つの変数を 2 つの比較演算子の間に指定し、上限と下限の両方を指定して構成します。たとえば、次の式では、500 - 1000 の範囲内の従業員番号のオブザベーションが選択されます。この範囲には、500 および 1000 は含まれます。

```
where 500 <= empnum <= 1000;
```

上の例の範囲指定条件式は、次の式と同じ意味を持ちます。

```
where empnum >= 500 and empnum <= 1000;
```

NOT 論理演算子を、範囲の指定と組み合わせることができます。この組み合わせにより、指定した範囲の外にあるオブザベーションを選択できます。この場合、次の例のようにかっこが必要です。

```
where not (500 <= empnum <= 1000);
```

BETWEEN-AND 演算子

BETWEEN-AND 演算子も、上限と下限が指定された範囲条件と見なされ、変数の値が指定した範囲内にあるオブザベーションを選択します。

範囲の上限および下限は、定数または式として指定します。指定された範囲には、上限の値および下限の値も含まれます。BETWEENAND 演算子を使用するための一般的な構文は次のとおりです。

WHERE variable BETWEEN value AND value;

次に例を示します。

```
where empnum between 500 and 1000;
where taxes between salary*0.30 and salary*0.50;
```

NOT 論理演算子を、BETWEEN-AND 演算子と組み合わせることができます。この組み合わせにより、指定した範囲の外にあるオブザベーションを選択できます。

```
where empnum not between 500 and 1000;
```

注: BETWEENAND 演算子により得られる結果と、範囲の指定により得られる結果は同じです。つまり、次の 2 つの WHERE 式は同じ意味を持ちます。

```
where 500 <= empnum <= 1000;
where empnum between 500 and 1000;
```

CONTAINS 演算子

CONTAINS 演算子(?演算子)の一般的な用途は、指定された文字列を含む文字変数の値を検索してオブザベーションを選択することです。変数値内での文字列の位置は処理に影響しません。ただし、CONTAINS 演算子では、アルファベットの大文字と小文字は区別して比較します。

次の例では、変数 Company の値が **Mobay** および **Brisbayne** であるオブザベーションが選択されます。しかし、値が **Bayview** のオブザベーションは、この例での選択の対象外です。

```
where company contains 'bay';
where company ? 'bay';
```

NOT 論理演算子を、CONTAINS 演算子と組み合わせることができます。この組み合わせにより、指定した文字列が含まれないオブザベーションを選択できます。

```
where company not contains 'bay';
```

また、CONTAINS 演算子では、2 つの変数を使用できます。これにより、一方の変数がもう一方の変数に含まれているかどうかを調べることができます。2 つの変数を指定する場合は、後置ブランクが含まれる可能性があります。後置ブランクは、TRIM 関数を使用して削除します。

```
proc sql;
   select *
   from table1 as a, table2 as b
   where a.fullname contains trim(b.lastname) and
      a.fullname contains trim(b.firstname);
```

また、TRIM 関数はマクロ変数を検索する場合にも有用です。

```
proc print;
   where fullname contains trim("&lname");
run;
```

IS NULL 演算子または IS MISSING 演算子

IS NULL 演算子または IS MISSING 演算子は、変数の値が欠損しているオブザベーションを選択します。これらの演算子は、欠損値または特殊欠損値の文字のあるオブザベーションを選択します。これらの演算子は文字変数と数値変数の両方に使用できます。

```
where idnum is missing;
where name is null;
```

文字データの場合、次の式は同じ意味を持ちます。

```
where name is null;
where name = ' ';
```

また、数値データの場合、次の式は同じ意味を持ちます。次のステートメントは、欠損値が特殊欠損値である文字を区別します。

```
where idnum <= .Z;
```

NOT 論理演算子を、IS NULL 演算子または IS MISSING 演算子と組み合わせることにより、非欠損値を選択できます。

```
where salary is not missing;
```

LIKE 演算子

LIKE 演算子は、文字変数の値と指定したパターンの比較によってオブザベーションを選択します。これは、パターンマッチングと呼びます。LIKE 演算子では、アルファベットの大文字と小文字が区別されます。次の 2 種類の特殊文字が、パターンの指定に使用できます。

パーセント(%)
: 任意の数の文字がその位置にあることを表します。次の WHERE 式では、名前が **N** の文字で始まる従業員がすべて選択されます。名前の長さは任意です。

  ```
  where lastname like 'N%';
  ```

アンダースコア(_)
: 値の中の 1 文字に一致します。1 つのアンダースコア(_)は 1 つの文字を表します。1 つのパターンの中に、複数のアンダースコア(_)を連続して指定できます。また、同一のパターンの中には、パーセント(%)とアンダースコア(_)の両方を指定できます。たとえば、次のような名前のリストがあるとします。

  - **Diana**
  - **Diane**
  - **Dianna**
  - **Dianthus**
  - **Dyan**

次の表は、さまざまなパターンの形式で LIKE 演算子を使用した場合に、それによって、どのような名前が選択されるかを示しています。

パターン	選択される名前
like 'D_an'	Dyan
like 'D_an_'	Diana、Diane
like 'D_an__'	Dianna
like 'D_an%'	リストに含まれるすべての名前

パターンは、SAS 文字式を使用して指定できます。ただし、SAS 関数を使用する SAS 文字式は除きます。

NOT 論理演算子を、LIKE 演算子と組み合わせることにより、次の例のように、指定したパターンと一致しない値を選択できます。

```
where frstname not like 'D_an%';
```

LIKE 演算子の場合、%文字と_文字には特殊な意味があるので、値に含まれる%文字や_文字を検索する場合はエスケープ文字を使用する必要があります。エスケープ文字は、文字の連なりの中で、次に来る文字が代わりの意味を持つことを示す 1 文字です。LIKE 演算子の場合、エスケープ文字は、特殊文字関数を実行するかわりに、変数値に含まれる%文字や_文字のリテラルインスタンスを検索することを示します。

たとえば、変数 X に **abc**、**a_b** および **axb** が含まれている場合、エスケープ文字を付けた次の LIKE 演算子は **a_b** の値のみを選択します。エスケープ文字(/)は、パターンが **a** および **b** という文字の間のリテラル' _'を検索することを示します。エスケープ文字(/)は検索対象には含まれません。

```
where x like 'a/_b' escape '/';
```

エスケープ文字を付けなかった場合、次の LIKE 演算子は **a_b** と **axb** を選択します。検索パターン内の特殊文字アンダースコアは、アンダースコア付きの値を含め、任意の **b** 文字 1 つとマッチします。

```
where x like 'a_b';
```

エスケープ文字を指定するには、パターンマッチング式に文字を含め、その次にキーワード ESCAPE、エスケープ文字式を入れます。エスケープ文字を含める場合は、パターンマッチング式を引用符で囲み、列名が含まれないようにします。エスケープ文字式の評価は 1 文字です。オペランドは文字または文字列リテラルにする必要があります。1 文字の場合は、引用符で囲みます。

```
LIKE 'pattern-matching-expression' ESCAPE 'escape-character-expression'
```

SOUNDS-LIKE 演算子

SOUNDS-LIKE (=*)演算子は、指定した単語または語句のスペルに類似する文字列が含まれるオブザベーションを選択します。この演算子では、Soundex アルゴリズムにより変数値とオペランドを比較します。詳細については、*SAS Functions and CALL Routines: Reference* の SOUNDEX 関数を参照してください。

注: SOUNDEX アルゴリズムは英語を主な対象としており、英語以外の言語では利便性が低くなります。

=*(SOUNDS-LIKE)演算子は有用です。しかし、想定される値のすべてが選択されるとは限らないことに注意してください。たとえば、次のリストから、名前が Smith に類似するオブザベーションを選択する場合を考えます。

- **Schmitt**
- **Smith**
- **Smithson**
- **Smitt**
- **Smythe**

次の WHERE 式では、このリストから **Smithson** を除く名前がすべて選択されます。

```
where lastname=* 'Smith';
```

NOT 論理演算子を、=*(SOUNDS-LIKE)演算子と組み合わせることができます。この組み合わせにより、指定した単語または語句のスペルに類似する文字列が含まれない値を選択できます。

```
where lastname not =* 'Smith';
```

注: =*(SOUNDS-LIKE)演算子は、インデックスにより最適化することはできません。

SAME-AND 演算子

SAME-AND 演算子は、プログラム内の既存の WHERE 式に、条件を再入力しないで追加することができます。この機能は、次の場合に役立ちます。

- 対話型モードの SAS プロシジャ実行時
- WHERE 式をコマンドラインに入力可能な SAS ウィンドウ環境
- RUN グループ処理時

SAME-AND 演算子を使用して定義済の WHERE 式に条件を追加します。SAME-AND 演算子の形式は次のとおりです。

- where-expression-1;
- ...*SAS statements*...
- WHERE SAME AND where-expression-2;
- ...*SAS statements*...
- WHERE SAME AND where-expression-n;

選択されるオブザベーションは、すでに定義されている条件に加えて、SAMEAND 演算子の後に指定した条件を満たします。条件はすべて、単一の WHERE 式の中の AND 演算子で区切られた条件であるかのように処理されます。

次の例は、GPLOT プロシジャにおける RUN グループ内で SAMEAND 演算子を使用する方法を示しています。SAS データセット YEARS には、3 つの変数が存在し、2009 年 - 2011 年の期間の、四半期単位のデータが含まれています。

```
proc gplot data=years;
   plot unit*quar=year;
run;

   where year > '01jan2009'd;
run;

   where same and year < '01jan2012'd;
run;
```

次の WHERE 式は、前述のコードと同じ意味を持ちます。

```
where year > ''01jan2009'd and year < '01jan2012'd;
```

MIN 演算子と MAX 演算子

2 つの数量の最小値または最大値を見つけるには、MIN 演算子または MAX 演算子を使用します。これらの演算子は、最小値または最大値を調べる 2 つの数値で囲んで指定します。

- 最小演算子(MIN)は、2 つの値を比較して小さい方の値を返します。
- 最大演算子(MAX)は、2 つの値を比較して大きい方の値を返します。

たとえば、A が B より小さい場合、次の式では A の値が返されます。

```
where x = (a min b);
```

注: 記号><は、WHERE 式ではサポートされない記号です。記号<>は、WHERE 式では不等号として解釈されます。

連結演算子

連結演算子は、文字値を連結します。連結演算子は、次のように指定します。

- || (実線の縦棒 2 つ)

- !!(感嘆符 2 つ)
- ¦¦ (切れ目のある縦棒 2 つ)

次に例を示します。

```
where name = 'John'||'Smith';
```

接頭演算子

正符号(＋)および負符号(－)は、接頭演算子として使用される場合と、算術演算子として使用される場合があります。接頭演算子となるのは、正符号(＋)または負符号(－)が式の先頭にある場合、または開きかっこの直前にある場合です。接頭演算子は、変数、定数、SAS 関数、かっこ付きの式に対して適用されます。

```
where z = -(x + y);
```

注: NOT 演算子は、接頭演算子でもあります。

論理演算子を用いた式の結合

構文

論理演算子を使用して、複数の WHERE 式を組み合わせて、複合 WHERE 式を作成することができます。論理演算子は、ブール演算子とも呼ばれ、その種類には AND、OR、NOT があります。複合 WHERE 式の基本的な構文は次のとおりです。

WHERE *where-expression-1* AND | OR | NOT *where-expression-n*

AND は、2 つの条件の両方を満たすオブザベーションを検索するように条件を組み合わせます。次に例を示します。

```
where skill eq 'java' and years eq 4;
```

OR は、2 つの条件の一方または両方を満たすオブザベーションを検索するように条件を組み合わせます。次に例を示します。

```
where skill eq 'java' or years eq 4;
```

NOT は、指定した条件の補集合を検索するように条件を組み合わせます。NOT 論理演算子は、任意の SAS 演算子および WHERE 式の演算子と組み合わせて使用できます。また、NOT 演算子は、AND 演算子や OR 演算子と組み合わせることもできます。次に例を示します。

```
where skill not eq 'java' or years not eq 4;
```

論理演算子の記号と対応するニーモニックを次の表に示します。

表 11.4 *論理(ブール)演算子*

記号	ニーモニック
&	AND
! 、|、¦	OR
^ 、~ 、¬	NOT

複合式の処理

複合 WHERE 式(複数の条件)が検出されると、規則に従って各式の評価順序が決定されます。WHERE 式が組み合わされている場合、条件は次の順序で処理されます。

1. 最初に、NOT 式が処理されます。
2. 次に、AND で連結された式が処理されます。
3. 最後に、OR で連結された式が処理されます。

かっこを用いた評価の順序の管理

論理演算子は特定の順序で評価されますが、式をかっこで囲んでネストすることによって、評価順序を制御できます。つまり、かっこで囲まれた式が、囲まれていない式よりも先に処理されます。最初に、最も内側のかっこで囲まれた式が処理されます。次に、その 1 つ外側のかっこで囲まれた式が処理されます。さらに、外側へ向けて、すべてのかっこが処理されるまで処理が継続されます。

たとえば、SAS/GRAPH と SAS/STAT のソフトウェアを両方とも所有しているカナダのサイトすべてのリストを取得する場合を考えます。次のような WHERE 式を指定してみます。

```
where product='GRAPH' or product='STAT' and country='Canada';
```

しかし、この式の結果には、SAS/GRAPH ソフトウェアのライセンスを所有するすべてのサイトと、SAS/STAT ソフトウェアのライセンスを所有するカナダのサイトが含まれます。正しい結果を得るには、次のようにかっこを使用します。かっこを使用することにより、かっこ内の比較が最初に評価されます。これによって、いずれかのプロダクトライセンスを所有しているサイトのリストがまず取得され、その取得される結果が残りの条件で使用されます。

```
where (product='GRAPH' or product='STAT') and country='Canada';
```

WHERE 処理のパフォーマンスの改善

SAS データセットにインデックスを作成すると、WHERE 処理のパフォーマンスが大幅に向上します。インデックスとは、SAS データファイル用に作成するオプション指定のファイルです。インデックスを使用すると、特定のオブザベーションに直接アクセスできます。

インデックスを使用しないで WHERE 式を処理する場合、オブザベーションが順番に読み込まれて、選択条件に一致するものが検索されます。インデックスを使用しない場合、前の SORT プロシジャまたは SORTEDBY=データセットオプションからのデータファイルで格納されているソートインジケータが最初に確認されます。ソートインジケータが有効な場合、SAS はそれを利用し、WHERE 式を満たす値がそれ以上ないことが明らかになると、ファイルの読み込みを停止します。たとえば、インデックスを使用せずに年齢で並べ替えるデータセットがあるとします。式 **where age le 25** を処理する場合、SAS は、25 を超えるオブザベーションが見つかると、オブザベーションの読み込みを停止します。SAS がオブザベーションの読み込みを停止するタイミングを判別できる場合、インデックスを使用しなければ、どこで始めるかを示す情報がありません。そのため、SAS は最初のオブザベーションから常に始まり、多くのオブザベーションを読み込む可能性があります。

インデックスを使用すると、SAS は、条件を満たすオブザベーションを判別できます。これは、WHERE 式を最適化するものと見なされます。デフォルトでは、インデックスを使用するか、または、データセット全体を順番に読み込むかについては、SAS System が決定します。SAS が WHERE 式を処理するためのインデックスを使用する方法の詳細については、“WHERE 処理でのインデックスの使用” (642 ページ)を参照してください。

次の表では、データセットのインデックスを作成する以外の、効率のよい WHERE 式を作成するためのガイドラインを紹介します。

表 11.5 *効率のよい WHERE 式の作成*

ガイドライン	効率的	非効率的
%または_で始まる LIKE 演算子を使用しない。	`where country like 'A%INA';`	`where country like '%INA';`
算術演算式を使用しない。	`where salary > 48000;`	`where salary > 12*4000;`

条件付きで選択されたデータセグメントの処理

FIRSTOBS=オプションと OBS=オプションの適用

WHERE 式を使用して条件に基づいてオブザベーションのサブセットを選択する場合、FIRSTOBS=または OBS=、あるいはその両方の処理を(データセットオプションおよびシステムオプションとして)適用することで、そのサブセットをセグメント化することもできます。WHERE 式で使用する場合、

- FIRSTOBS=は、処理を開始するために WHERE 式で選択したデータのサブセット内のオブザベーション番号を指定します。
- OBS=は、WHERE 式で選択したデータのサブセットからオブザベーションの処理を中止するタイミングを指定します。

WHERE 式で使用する場合、OBS=および FIRSTOBS=用に指定した値は、データセットの物理オブザベーション番号ではなく、サブセット内の論理番号です。たとえば、`obs=3` は、データセット内の 3 番目のオブザベーション番号ではなく、WHERE 式で選択したデータのサブセットの 3 番目のオブザベーションのことです。

データのサブセットに OBS=および FIRSTOBS=処理を適用することは、SQL プロシジャの WHERE ステートメント、WHERE=データセットオプション、WHERE 句でサポートされています。

別のビューのビュー(ネストされたビュー)である SAS ビューを処理する場合、OBS=と FIRSTOBS=をデータのサブセットに適用すると、予期しない結果になる可能性があります。ネストされたビューの場合、OBS=および FIRSTOBS=処理が、ルート(最下位レベル)ビューから始まり、SAS ビューごとにオブザベーションをフィルタリングして、各 SAS ビューに適用されます。その結果、サブセットとセグメントの条件に合うオブザベーションがない可能性もあります。“SAS ビューの処理” (179 ページ).を参照してください。

データのサブセットに FIRSTOBS=と OBS=を適用する

次の SAS プログラムでは、サブセットデータの条件を指定する方法、および処理対象データのサブセットのセグメントを指定する方法を示します。

```
data A; 1
   do I=1 to 100;
   X=I + 1;
   output;
   end;
run;

proc print data=work.a (firstobs=2 3 obs=4; 4
   where I > 90; 2
run;
```

1 DATA ステップは、100 のオブザベーションと 2 つの変数(I と X)を含む Work.A というデータセットを作成します。

2 WHERE 式 `I > 90` では、指定した条件に合致するオブザベーションのみを処理するように SAS に指示します。これにより、オブザベーション 91 - 100 のサブセットが一致します。

3 FIRSTOBS=データセットオプションは、データのサブセットの 2 番目のオブザベーションで処理を開始するように SAS に指示します。ここでは、オブザベーション 92 が一致します。

4 OBS=データセットオプションは、データのサブセットの 4 番目のオブザベーションに達したら処理を停止するように SAS に指示します。ここでは、オブザベーション 94 が一致します。

PROC PRINT の結果はオブザベーション 92、93、94 です。

SAS ビューの処理

次の SAS プログラムでは、データセット、そのデータセットの SAS ビュー、最初の SAS ビューを基にデータをサブセット化する 2 番目の SAS ビューを作成します。WHERE ステートメントと OBS=システムオプションの両方が使用されます。

```
data a; 1
   do I=1 to 100;
   X=I + 1;
   output;
   end;
run;

data viewa/view=viewa; 2

   set a;
      Z = X+1;
run;

data viewb/view=viewb; 3

   set viewa;
      where I > 90;
run;
```

```
options obs=3; 4

proc print data=work.viewb; 5

run;
```

1 最初の DATA ステップは、100 個のオブザベーションと 2 つの変数(I と X)を含む Work.A というデータセットを作成します。

2 2 番目の DATA ステップは、100 個のオブザベーションと I、X (データセット Work.A から)、Z (この DATA ステップで割り当て)という 3 つの変数を含む Work.ViewA という SAS ビューを作成します。

3 3 番目の DATA ステップは、Work.ViewB という SAS ビューを作成し、WHERE ステートメントでデータをサブセット化します。これにより、10 個のオブザベーションにアクセスするビューが一致します。

4 OBS=システムオプションは、前の SET ViewA ステートメントに適用されます。このステートメントは、処理されているデータのサブセットの 3 番目のオブザベーションに達すると処理を停止するように SAS に指示します。

5 PRINT プロシジャが処理された場合、次の処理が実行されます。

1. 最初に、SAS は *obs=3* を Work.ViewA に適用します。これにより、3 番目のオブザベーションで処理が停止します。
2. 次に、SAS は条件 *I > 90* を、処理対象となる 3 つのオブザベーションに適用します。条件に一致するオブザベーションは存在しません。.
3. PROC PRINT には、結果的にオブザベーションがありません。.

予期しない結果が発生する可能性を防ぐには、ルート(最下位レベル)ビューのすべてのオブザベーションを強制的に読み込むために、Work.ViewA を作成する際に **obs=max** を指定します。

```
data viewa/view=viewa;
   set a (obs=max);
   Z = X+1;
run;
```

PRINT プロシジャはオブザベーション 91、92、93 を処理します。

WHERE 式とサブセット化 IF ステートメントの使い分け

SAS データセットからある条件に基づいてオブザベーションを選択するには、WHERE 式、または、サブセット化 IF ステートメントを使用します。どちらも、条件をテストして、SAS がオブザベーションを処理するかどうかを決定します。ただし、次のような違いがあります。

- サブセット化 IF ステートメントは、DATA ステップでのみ使用できます。サブセット化 IF ステートメントでは、条件がテストされるのは、オブザベーションが PDV (プログラムデータベクトル)に読み込まれた後です。条件が真である場合は、現在のオブザベーションに対する処理が継続されます。条件が偽である場合は、そのオブザベーションが破棄されて、次のオブザベーションが処理されます。
- WHERE 式は、DATA ステップと SAS プロシジャの両方で使用できます。また、ウィンドウ環境、SCL プログラム、データセットオプションでも使用することができます。WHERE 式では、オブザベーションが PDV に読み込まれる前に条件がテスト

されます。条件が真である場合は、オブザベーションが PDV に読み込まれて処理されます。条件が偽である場合は、オブザベーションは PDV に読み込まれず、次のオブザベーションに処理が継続します。これは、オブザベーションが多くの変数を含む場合や、長さの大きい文字変数(最大で 32KB)を含む場合には、非常に有効です。さらに、WHERE 式は、インデックスを使用して最適化でき、その上、LIKE 演算子や CONTAINS 演算子のような特殊な演算子も指定できます。

注: 一般的に、WHERE 式を使用する方が、PDV に読み込まれる前に処理されるため、より効率的です。ただし、変数の個数が少ないオブザベーションを含むデータセットでは、PDV に読み込むことによる効率への影響は小さいと言えます。一方、たとえば、1 つの変数に 32KB の長い文字データを含む場合は、変数の個数が 1 つであっても、効率への影響は小さくないと言えます。

ほとんどの場合、どちらの方法も使用できます。ただし、どちらか 1 つの方法を使用しなければならないタスクもあります。次の表に主なタスクの違いを示します。

表 11.6 *WHERE 式またはサブセット化 IF ステートメントが必要なタスク*

タスク	方法
DATA ステップを使用せずに、PROC ステップ内で選択する場合。	WHERE 式
効率を重視して、インデックス付きデータセットを使用する場合。	WHERE 式
BETWEEN-AND、CONTAINS、IS MISSING、IS NULL、LIKE、SAME-AND、=*(SOUNDS-LIKE)などの特殊な演算子を使用する場合。	WHERE 式
SAS データセット中の変数以外の値に基づいて選択する場合。たとえば、生データから読み込まれる値や、DATA ステップの実行中に計算される値または割り当てられる値などがあります。	サブセット化 IF
DATA ステップの先頭ではなく、実行中に選択する場合。	サブセット化 IF
選択処理を条件付きで実行する場合。	サブセット化 IF

12 章

システムパフォーマンスの最適化

システムパフォーマンスの最適化の定義

パフォーマンス統計量
: パフォーマンス統計量とは、個々の DATA ステップや PROC ステップの処理に使用される、入出力操作(I/O)、メモリ、CPU 時間の統計量です。この統計量は、SAS システムオプションを使用して取得することができ、SAS ジョブの初期パフォーマンスを測定してパフォーマンスを向上させるための方法を決定するのに役立ちます。

システムパフォーマンス
: SAS プログラムの処理に使用されるシステムの入出力、メモリ、CPU 時間の統計量です。次に説明するいくつかの手法を使用して、これらの重要なリソースを節約したり、割り当て直したりすることにより、システムパフォーマンスを向上させることができます。状況によっては期待通りにならない場合もありますが、状況に見合う適切な手法を選択してください。

パフォーマンス統計量の収集と解釈

FULLSTIMER システムオプションと STIMER システムオプションの使用

FULLSTIMER および STIMER システムオプションを使用すると、パフォーマンス統計量を SAS ログに出力するかどうかを制御できます。これらのシステムオプションで生成される結果は、動作環境によって異なります。これらのシステムオプションで生成される出力の詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

次の出力結果は、UNIX 動作環境における SAS ログ内の FULLSTIMER システムオプションの出力の例です。

ログ 12.1 *FULLSTIMER システムオプションを UNIX 動作環境で使用したサンプル結果*

```
NOTE:DATA statement used:real time 0.19 seconds user cpu time 0.06 seconds
system cpu time 0.01 seconds Memory 460k Semaphores exclusive 194 shared 9
contended 0 SAS Task context switches 1 splits 0
```

STIMER システムオプションを指定すると、FULLSTIMER 統計量のサブセットがレポートされます。SAS ログ内における STIMER システムオプションの出力結果の例を次に示します。

ログ 12.2 *STIMER システムオプションを UNIX 動作環境で使用したサンプル結果*

```
NOTE:DATA statement used: real time           1.16 seconds cpu time
0.09 seconds
```

動作環境の情報
: 各動作環境における STIMER システムオプションと FULLSTIMER システムオプションの違いについては、各動作環境に対応するドキュメントを参照してください。これらのオプションで表示される情報は、動作環境によって異なります。このため、表示される統計量は、サンプル結果と異なる場合があります。

FULLSTIMER 統計量と STIMER 統計量の解釈

STIMER および FULLSTIMER システムオプションを使用すると、処理時間(経過時間)や CPU 時間など、数種類のリソース使用統計量がレポートされます。処理時間は、ジョブやステップの実行にかかった時間を表し、システムの容量および負荷によって大幅に変動します。特定のリソースを共有しているユーザーが多いほど、1 ユーザーが利用できるリソースは少なくなります。CPU 時間は、ジョブの実行にかかった CPU の実際の処理時間を表します。システムの容量および負荷による影響はありません。リソースの待機時間が長くなる場合、CPU 時間は増加しませんが、処理時間は増加します。プログラムの効率を処理時間のみで判断することはできません。その理由は、システムの容量や負荷要求をユーザーが任意に制御できないからです。システムパフォーマンスをより正確に評価するには、CPU 時間を使用します。プログラムを修正して効率を上げると、CPU 時間はほぼ予測したとおりに減少します。

FULLSTIMER システムオプションでレポートされる統計量は、入出力、メモリ、CPU 時間の 3 つの重要なコンピュータリソースに関連しています。多くの状況では、これらの、どのリソースの使用量を減らしても、特定の SAS ジョブのスループットが向上し、処理時間が減少します。ただし、以下で説明するように、例外もあります。

I/O 最適化の手法

I/O 最適化の手法の概要

入出力操作は、パフォーマンスを最適化するために最も重要な要素の 1 つです。ほとんどの SAS ジョブでは、特定のデータセットを読み込み、各種のデータ分析やデータ操作のタスクを実行するという処理が繰り返し行われます。SAS ジョブのパフォーマンスを向上させるには、SAS System がディスクデバイスやテープデバイスにアクセスする回数を減らす必要があります。

そのためには、次の方法によって、必要な変数やオブザベーションだけが処理されるように SAS プログラムを変更します。

- WHERE 処理の使用
- DROP ステートメントと KEEP ステートメントの使用
- LENGTH ステートメントの使用
- OBS=データセットオプションと FIRSTOBS=データセットオプションの使用

また、次の方法によって、データの内部的な処理回数を減らすようにプログラムを変更することもできます。

- SAS データセットを作成する
- インデックスを使用する
- SAS ビューを通じてデータにアクセスする
- エンジンを効率的に使用する
- 変数属性の変更時に PROC DATASETS を使用する
- 数値を文字として保存する
- メモリ使用量を最適化する手法を使用する

デバイスへの 1 回のアクセスで処理されるデータの量を増やし、データアクセスの回数を減らすには、次の操作を行います。

- ALIGNSASIOFILES、BUFNO=、BUFSIZE=、CATCACHE=、COMPRESS=、DATAPAGESIZE=、STRIPESIZE=、UBUFNO=、UBUFSIZE=システムオプションを設定する
- SASFILE グローバルステートメントを使用して SAS データセットを開き、データセット全体をメモリに保持できるバッファを割り当てる

SAS DATA ステップビューの使用時に、パフォーマンスを向上させるには次の操作を行います。

- VBUFSIZE=システムオプションを指定する
- OBSBUF=データセットオプションを指定する

注: 複数の方法を組み合わせて使用することで、SAS ジョブの効率がさらに向上する場合もあります。

WHERE 処理の使用

プロシジャで WHERE ステートメントを使用すると、サブセット化 IF ステートメントを使用する DATA ステップと同じタスクを実行できます。WHERE ステートメントを使用すると、不要なオブザベーションが処理されないので、特定の分析を実行するときに、余分な DATA ステップ処理を省略することができます。

たとえば、次の DATA ステップでは、データセット Seatbelt が作成されます。このデータセットには、Auto.Survey データセットで、変数 Seatbelt の値が **YES** であるオブザベーションのみが含まれます。続いて、作成されたデータセットが出力されます。

```
libname auto 'SAS-library';
data seatbelt;
   set auto.survey;
   if seatbelt='yes';
run;

proc print data=seatbelt;
run;
```

一方、PRINT プロシジャで WHERE ステートメントを使用した場合は、データセットを作成しなくても、PROC PRINT ステップから同じ出力結果が得られます。例を次に示します。

```
proc print data=auto.survey;
   where seatbelt='yes';
run;
```

WHERE ステートメントを使用すると、データ処理の回数が少なくなるので、リソースを節約することができます。この例では、DATA ステップが省略されているので、CPU 時間とメモリを節約することができます。また、中間データセットがないので、入出力の回数も少なくて済みます。ただし、生データを読み込む DATA ステップでは、WHERE ステートメントを使用できません。

リソースをどれだけ節約できるかは、データセットのサイズを含め、多くの要因によって異なります。いろいろな手法をプログラムで試し、最も効率的なものを実際に使用することをお勧めします。詳細については、"WHERE 式とサブセット化 IF ステートメントの使い分け" (180 ページ)を参照してください。

DROP ステートメントと KEEP ステートメントの使用

パフォーマンスの効率を高めるためのもう 1 つの方法は、DROP または KEEP ステートメントを使用して、オブザベーションのサイズを小さくすることです。一時データセット

を作成して、必要な変数だけが含まれるようにすると、データ処理に必要な入出力操作の回数を減らすことができます。詳細については、“DROP Statement” (*SAS Statements: Reference*)および “KEEP Statement” (*SAS Statements: Reference*)を参照してください。

LENGTH ステートメントの使用

LENGTH ステートメントを使用することでも、オブザベーションのサイズを縮小できます。それぞれの変数に必要な内容だけが含まれるように変数の長さを変更すると、データ処理に必要な入出力操作の回数を減らすことができます。ただし、数値変数の長さを変更する前に、“変数の長さの指定” (193 ページ)を確認しておいてください。詳細については、“LENGTH Statement” (*SAS Statements: Reference*)を参照してください。

OBS=データセットオプションと FIRSTOBS=データセットオプションの使用

OBS=および FIRSTOBS=データセットオプションを使用することでも、処理するオブザベーションの数を削減できます。一時データセットを作成して、必要なオブザベーションだけが含まれるようにすると、データ処理に必要な入出力操作の回数を減らすことができます。詳細については、“FIRSTOBS= Data Set Option” (*SAS Data Set Options: Reference*)および“OBS= Data Set Option” (*SAS Data Set Options: Reference*)を参照してください。

SAS データセットの作成

同じ生データを何回も処理する場合は、通常、SAS データセットを作成した方が効率的です。SAS System は、生データファイルよりも、SAS データセットをより効率的に処理できます。

以前のバージョンの SAS ソフトウェアで作成されたデータセットを使用しているかどうかも、検討の対象となります。以前のバージョンで作成されたデータセットを頻繁に処理するよりも、最新のバージョンの SAS ソフトウェアを使用して、新しいデータセットに作成し直す方が効率的です。詳細については、33 章, “SAS 9.4 における、以前のリリースの SAS ファイルとの互換性” (717 ページ) を参照してください。

インデックスの使用

インデックスとは、SAS データファイル用に作成するオプション指定のファイルです。インデックスを使用すると、特定のオブザベーションに直接アクセスできます。インデックスは、特定の変数の値を昇順で格納するほか、データファイルのオブザベーション内の値の場所に関する情報を含んでいます。つまり、インデックスを使用すると、インデックス化された変数の値でオブザベーションを検索できます。

インデックスがない場合、SAS System は、データファイル内部の格納順にオブザベーションにシーケンシャルアクセスします。インデックスがある場合、SAS System はオブザベーションに直接アクセスします。そのため、インデックスを作成して使用すると、オブザベーションへのアクセスが速くなります。

一般に、SAS System でインデックスを使用すると、次の状況においてパフォーマンスを向上させることができます。

- WHERE 式の処理でインデックスを使用すると、データのサブセットに、より高速かつより効率的にアクセスできます。
- BY グループ処理でインデックスを使用すると、SORT プロシジャを使用しなくても、オブザベーションがインデックス順(値の昇順)に返されます。

- SET ステートメントと MODIFY ステートメントの場合、KEY=オプションを使用することによって、DATA ステップでインデックスを指定して、データファイルの特定のオブザベーションを取得することができます。

注: インデックスを作成することにより、処理効率は向上します。しかし、インデックスは、一部のリソースを保護する一方で、その他のリソースを消費します。したがって、インデックスの作成、使用、管理に関連するリソース効率について考慮する必要があります。インデックスの詳細およびインデックスを作成するかどうかを判断する材料については、“SAS インデックスについて” (632 ページ)を参照してください。

SAS ビューを通じたデータアクセス

SQL プロシジャまたは DATA ステップを使用して、データの SAS ビューを作成することができます。SAS ビューとは、データ値やディスクリプタ情報の取り出しに必要な情報だけが物理的に格納されている SAS データセットです。SAS ビューを使用すると、より少ないステートメントでデータをサブセット化することができます。また、SAS ビューを使用して、複数のデータセットに含まれるデータをグループ化することもできます。この場合も、新しいデータセットを作成する必要がないので、処理時間とディスク領域の両方を節約することができます。詳細については、27 章, “SAS ビュー” (665 ページ) および *Base SAS Procedures Guide* を参照してください。

SAS ビューを使用したシステムパフォーマンスの最適化の詳細については、“SAS DATA ステップビューに対する VBUFSIZE=と OBSBUF=の設定” (191 ページ)を参照してください。

エンジンの効率的な使用

LIBNAME ステートメントでエンジンを指定しない場合、SAS ライブラリに関連付けるエンジンを特定するために余分な処理ステップを実行する必要があります。使用するエンジンを特定できる情報が見つかるまで、ディレクトリ内のファイルがすべて確認されます。たとえば、次のステートメントでは、ライブラリ参照名 Fruits で特定のエンジンを使用することが明示的に指示されているため効率的です。

```
/* Engine specified. */

libname fruits v9 'SAS-library';
```

次のステートメントでは、エンジンが明示的に指定されていません。出力結果では、エンジンの種類が特定できないことを示す NOTE メッセージが出力されます。

```
/* Engine not specified. */

libname fruits 'SAS-library';
```

ログ 12.3 *LIBNAME ステートメントの SAS ログ出力*

```
NOTE:Directory for library FRUITS contains files of mixed engine
types.NOTE:Libref FRUITS was successfully assigned as follows:Engine:       V9
Physical Name:SAS-library
```

z/OS 固有

z/OS 動作環境では、特定の種類のライブラリに対してエンジンを指定する必要がありません。

SAS エンジンの詳細については、35 章, “SAS エンジン” (737 ページ) を参照してください。

I/O パフォーマンス向上のためのシステムオプション設定

次の SAS システムオプションを使用すると、SAS ファイルで必要なディスクへのアクセス回数を減らすことができます。ただし、メモリの使用量と SAS データセットサイズが増大する可能性もあります。

ALIGNSASIOFILES
SAS データセットは、ヘッダーとその後に続く 1 ページ以上のデータから成ります。通常、ヘッダーは、Windows では 1K で UNIX では 8K です。
ALIGNSASIOFILES システムオプションでは、ヘッダーが強制的にデータページと同じサイズにされて、データページは境界に配置されるため、より効率的な I/O が可能になります。ページサイズは BUFSIZE=オプションを使用して設定されます。

詳細については、“ALIGNSASIOFILES System Option” (*SAS System Options: Reference*)、および使用している動作環境に対応する SAS ドキュメントを参照してください。

BUFNO=
BUFNO=システムオプションを使用して、SAS データセットの処理に必要なページバッファ数を調整することができます。このオプションの値を大きくすると、より少ないアクセス回数でデータを読み込むことができるので、アプリケーションのパフォーマンスが向上します。ただし、メモリの使用量は増大します。このオプションの値をさまざまに変更して、最適な値を使用するようにします。

注: CBUFNO=システムオプションを使用して、開いている SAS カタログに割り当てられる拡張ページバッファ数を制限することもできます。

詳細については、“BUFNO= System Option” (*SAS System Options: Reference*)、および使用している動作環境に対応する SAS ドキュメントを参照してください。

BUFSIZE=
BUFSIZE=システムオプションを使用して、BASE Engine がデータセットを作成するときの、データセットの永久ページサイズを指定することができます。ページサイズとは、1 回の入出力操作で 1 つのバッファに転送可能なデータ量のことです。BUFSIZE=のデフォルト値は、動作環境によって決まります。デフォルトでは、シーケンシャルアクセス方法を最適化するように設定されています。ダイレクト(ランダム)アクセスのパフォーマンスを向上させるには、BUFSIZE=の値を変更する必要があります。

ページバッファがいっぱいであっても空であっても、データセットは常にページバッファ単位で書き出されます。これは、動作環境のデフォルト値を使用した場合も、任意の値を指定した場合も同じです。

データの総量が小さいと見込まれる場合は、BUFSIZE=システムオプションで小さなバッファサイズを指定してもかまいません。データセットの大きさは小さいまま、ページバッファの無駄な領域を最小限にすることができます。一方、データセットのオブザベーション数が多くなると見込まれる場合は、BUFSIZE=システムオプションに、最適な値を使用して、オーバーヘッドを可能な限り小さくします。各ページバッファで追加のオーバーヘッドが必要になるので注意してください。

大きなデータセットにシーケンシャルアクセスする場合は、バッファサイズを大きくすると有利です。シーケンシャルアクセスでは、データセットの読み込みに必要なシステムの呼び出し回数が減ることによります。オブザベーションは複数のページにわたって存在することができないので、通常、ページには未使用領域が発生することに注意してください。

“データセットサイズの計算” (194 ページ) では、データセットサイズの推定方法について説明しています。

詳細については、“BUFSIZE= System Option” (*SAS System Options: Reference*)、および使用している動作環境に対応する SAS ドキュメントを参照してください。

CATCACHE=

CATCACHE=システムオプションを使用して、同時に開いておける SAS カタログ数を指定することができます。この値を大きくすると、メモリの使用量が増大します。アプリケーションで使用されているカタログが、他のアプリケーションでもすぐ必要となるような場合は、効果があります。これは、最初のアプリケーションで使用したカタログがキャッシュされるため、後続のアプリケーションのアクセスが効果的になるためです。

詳細については、“CATCACHE= System Option” (*SAS System Options: Reference*)、および使用している動作環境に対応する SAS ドキュメントを参照してください。

COMPRESS=

COMPRESS=システムオプションを使用すると、データセットを圧縮データセットとして格納するので、入出力処理の回数を減らすことができます。ただし、この方法でデータセットを格納した場合、利用時にオブザベーションを解凍しなければならないので、より多くの CPU 時間が必要になります。CPU 使用率よりも入出力のパフォーマンスを向上させたい場合は、この手法でデータセットを圧縮すると効果的です。

詳細については、“COMPRESS= System Option” (*SAS System Options: Reference*) を参照してください。

DATAPAGESIZE=

SAS 9.4 から、I/O パフォーマンス向上のため、最適なバッファページサイズが増やされています。ページサイズの増加によって、データセットまたはユーティリティファイルのサイズが増大する場合があります。使用している SAS セッションで現在の最適化プロセスが理想的とは考えられない場合、DATAPAGESIZE=COMPAT93 を指定して、SAS 9.4 より前に使用されていた最適化プロセスが使用できます。

詳細については、“DATAPAGESIZE= System Option” (*SAS System Options: Reference*)を参照してください。

STRIPESIZE=

データが RAID (Redundant Array of Independent Disks)デバイスに格納される場合、STRIPESIZE=システムオプションを使用すると、ディレクトリの I/O バッファサイズが RAID ストライプのサイズになるように設定できます。ディレクトリに作成される SAS データセットまたはユーティリティファイルのページサイズは、RAID ストライプサイズと一致します。このオプションを使用すると、個々のディスクのパフォーマンスが向上する可能性があります。

詳細については、“STRIPESIZE= System Option” (*SAS System Options: Reference*) を参照してください。

UBUFNO=

UBUFNO=システムオプションでは、データセットの処理に使用されるユーティリティバッファの数が設定されます。

詳細については、“UBUFNO= System Option” (*SAS System Options: Reference*)を参照してください。

UBUFSIZE=

UBUFSIZE=オプションでは、データセットの処理に使用されるユーティリティファイルのページサイズが設定されます。UBUFSIZE=オプションと BUFSIZE=オプションの値が同じ場合、ディスクへのアクセス回数が改善される可能性があります。

詳細については、“UBUFSIZE= System Option” (*SAS System Options: Reference*) を参照してください。

VBUFSIZE=
VBUFSIZE=オプションでは、ビューバッファのサイズが設定されます。多くの生成オブザベーションを保持するのに十分なビューバッファサイズを設定することにより、ビューのパフォーマンスを向上させられます。詳細については、“VBUFSIZE= System Option” (*SAS System Options: Reference*)および“SAS DATA ステップビューに対する VBUFSIZE=と OBSBUF=の設定” (191 ページ)を参照してください。

SAS DATA ステップビューに対する VBUFSIZE=と OBSBUF=の設定

SAS DATA ステップビューの処理時に、OBSBUF=データセットオプションか VBUSIZE=システムオプションのどちらかを指定すると、タスク切り替えが減少するので処理効率を向上させられます。VBUFSIZE=システムオプションでは、ビューバッファサイズをバイト数に基づいて指定できます。デフォルトバッファサイズは 65536 です。OBSBUF=データセットオプションでは、指定したオブザベーション数に基づいてビューバッファサイズが設定されます。いずれの場合も、より多くのオブザベーションを保持できるようにビューバッファを設定すると、タスク切り替えが減少して実行が高速化されます。

VBUFSIZE=システムオプションの詳細については、“VBUFSIZE= System Option” (*SAS System Options: Reference*)を参照してください。OBSBUF=データセットオプションの詳細については、“OBSBUF= Data Set Option” (*SAS Data Set Options: Reference*)を参照してください。

SASFILE ステートメントの使用

SASFILE グローバルステートメントを使用して SAS データセットを開き、データセット全体をメモリに保持できるバッファを割り当てます。データが読み込まれるとメモリに保持され、以降の DATA ステップと PROC ステップで利用できるようになります。これは、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、プログラムが終了してファイルが閉じられ、バッファが解放されるまで有効です。

SASFILE ステートメントを使用してパフォーマンスを向上させるには

- SAS データセットを処理するための複数のオープン/クローズ操作(バッファのメモリの割り当てと解放など)を 1 つのオープン/クローズ操作に減らします。
- データをメモリに保持して I/O 処理を軽減します。

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。また、SASFILE は、SAS/SHARE サーバーどの SAS サーバーを起動するプログラムの一部として特に役立ちます。SASFILE グローバルステートメントの詳細については、*SAS Statements: Reference* を参照してください。

DATASETS プロシジャを使用して属性を変更する

変数属性の変更が 1 つの PROC DATASETS 内で実施される唯一のタスクである場合、DATASETS プロシジャを使用して変数の属性を変更する方が、DATA ステップを使用するよりも効率的です。DATASETS プロシジャは、データセットのデータディスクリプタ情報のみを処理します。DATA ステップを実行すると、SAS データセット全体が処理されます。詳細については、“DATASETS” (*Base SAS Procedures Guide*)を参照してください。

変数を文字として保存

SAS System は、DATA ステップで処理される 1 つの数値につき 8 バイトの記憶域を、1 つの文字につき 1 バイトをそれぞれ使用します。数字を含んでいる変数に関して計算を実施する予定がない場合、その変数を文字変数として定義することにより記憶域を節約できます。各変数に必要となる記憶域の量を減らすことで、I/O 操作の回数を削減できます。

メモリ使用量に関する最適化手法

システムオプション

メモリが重要なリソースである場合、メモリの使用量を減らす手法はいくつかあります。ただし、ほとんどの場合、メモリの使用量を減らすと、入出力の回数や CPU 使用率は逆に増加します。

MEMSIZE=システムオプションを使用して利用可能なメモリ量を増やすことにより、処理時間を短縮できます。これは、メモリ量を増やすことにより、ページング(データのページをメモリに読み込むこと)にかかる時間が短くなるためです。

SORTSIZE=および SUMSIZE=システムオプションを使用すると、並べ替えおよび要約プロシジャで使用されるメモリ量を制限することができます。

“プログラムコンパイルの最適化を変更することにより CPU 時間を短縮” (194 ページ)で説明するように、メモリとその他のリソースとの間でトレードオフを行って処理時間を減らすこともできます。入出力サブシステムを最大限に活用するには、バッファ数とバッファサイズを増やす必要があります。ただし、バッファ領域は、SAS セッションの他の処理でも使用されます。

動作環境の情報

MEMSIZE=システムオプションは、すべての動作環境で使用できるわけではありません。また、特定の動作環境で MEMSIZE=システムオプションが使用できても、メモリの割り当てが増えない場合があります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

MEANS プロシジャで BY ステートメントを使用する

MEANS プロシジャで CLASS ステートメントとクラス変数を使用する場合、多くのメモリが必要となります。SAS System は、各クラス変数の一意の値のコピーをメモリ内に保持します。MEANS プロシジャですべての変数を要約しようするとメモリ不足になる場合、CLASS ステートメントと BY ステートメントを組み合わせて使用してクラス別にデータを分析することで、メモリを節約できます。

詳細については、“Comparison of the BY and CLASS Statements” (*Base SAS Procedures Guide*)を参照してください。

CPU パフォーマンスを最適化する手法

メモリの増加または I/O の削減により CPU 時間を短縮

ある一連のコードを実行するたびに、毎回ほぼ同じ CPU 時間がかかるとします。このような状況で CPU のパフォーマンスを最適化するには、通常、トレードオフが必要です。たとえば、使用するメモリを増やすと、CPU 時間を減らすことができます。これによ

り、1 回の操作で読み取りおよび格納される情報の量が増えます。ただし、他の処理で使用できるメモリは少なくなります。

入出力操作に必要な処理はすべて CPU で実行されるので、オプションや手法を使用して入出力操作の回数を減らすことにより、CPU のパフォーマンスを向上させることができます。

計算リソースを多用する DATA ステップの場合コンパイル済みプログラムを保存する

繰り返し実行される DATA ステップを、SAS ステートメントとしてではなく、コンパイル済みプログラムとして保存することによっても、CPU のパフォーマンスを向上させることができます。大きな DATA ステップで、入出力操作の回数が少ないジョブを処理する場合は、この手法が特に有効です。コンパイルされた DATA ステップの格納の詳細については、28 章, “コンパイル済みストアド DATA ステッププログラム” (675 ページ)を参照してください。

SAS 実行ファイルの検索時間の短縮

PATH=システムオプションを使用すると、SAS 実行ファイルが含まれるディレクトリ(一部の動作環境ではライブラリ)のリストを指定できます。デフォルトの環境設定ファイルには、このディレクトリの順序が指定されています。PATH=システムオプションでディレクトリの順序を変更することによって、アクセスの多いディレクトリを先に配置して指定することができます。また、アクセスの少ないディレクトリほど後ろに配置することもできます。

動作環境の情報
PATH=システムオプションを、使用できない動作環境もあります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

変数の長さの指定

プログラムデータベクトルの処理では、通常、個々の変数ではなく、1 つの大きな操作を単位としてデータを移動します。データが 8 バイトの境界に正しく整列している場合、データの移動は 2 クロック周期(1 回の読み取りに対して 1 回の格納)で行われます。データが整列していない場合は、より複雑な方法で移動します。最悪の場合、データは 1 回に 1 バイトずつ移動することになります。つまり、8 バイトの変数に対して、最低でも 8 倍の時間がかかることになります。

多くのハイパフォーマンスな RISC (Reduced Instruction Set Computer)プロセッサでは、未整列のデータを移動するとき、パフォーマンスが非常に大きく低下します。可能な限り、数値データは完全幅(8 バイト)にしておきます。算術演算を行う場合、数値データの幅が短い数値は、データの幅を大きくして指定する必要があります。一方、数値データの幅が短い数値は、メモリと入出力の両方を節約することができます。各動作環境や状況に応じて、どの方法が最も有利なのかを判断する必要があります。

注: 特定の変数を選択してデータセットを処理する場合や、WHERE の処理を使用する場合は、データの整列が特に重要になります。

並列処理の使用

SAS System 9 では、並列処理に関連する新しい SAS 機能をサポートしています。並列処理とは、複数の CPU で同時に処理することです。この技術では、SMP コンピュータを利用し、スレッド化 I/O とスレッド化アプリケーション処理という 2 種類の SAS プロセスのパフォーマンスゲインを提供します。

詳細については、13 章, “並列処理のサポート” (195 ページ)を参照してください。

プログラムコンパイルの最適化を変更することによりCPU 時間を短縮

SAS がプログラムをコンパイルする場合、配列サブスクリプトの冗長な指示、欠損値チェック、反復計算を削除するようにコードが最適化されます。コードは、指示のパターンを検出して、より効率的なシーケンスに置き換え、SAS レジスタに関する最適化を実行します。ほとんどの場合、コード生成最適化を実行することが望まれます。大きな DATA ステッププログラムがある場合、コード生成最適化を実行すると、コンパイル時間と実行時間が大幅に増加する可能性があります。

CGOPTIMIZE=システムオプションを使用すると、コード生成最適化を軽減するか無効にできます。次の CGOPTIMIZE=システムオプション値を使用して実行するコード生成最適化を設定します。

0 コードコンパイル時に最適化を実行しません。

1 ステージ 1 の最適化を実行するように指定します。ステージ 1 の最適化では、配列スクリプトの冗長な指示、欠損値チェック、反復計算を削除し、指示のパターンを検出し、より効率的なシーケンスに置き換えます。

2 ステージ 2 の最適化を実行するように指定します。ステージ 2 は、SAS レジスタに関連する最適化を実行します。ステージ 2 の最適化を大きな DATA ステッププログラムで実行すると、コンパイル時間が大幅に増える可能性があります。

3 ステージ 1 と 2 の組み合わせである完全な最適化を実行します。これはデフォルト値です。

詳細については、“CGOPTIMIZE= System Option” (*SAS System Options: Reference*)を参照してください。

データセットサイズの計算

パフォーマンス最適化のための手法を適用しても、まだ処理時間やメモリ使用率の問題が解決されない場合は、データセットのサイズが非常に大きい可能性があります。この場合、パフォーマンスをこれ以上向上させることは難しくなります。

処理に最適なデータセットのサイズを見積もるには、まず、データセットと同じ変数が含まれるダミーのデータセットを作成します。次に、CONTENTS プロシジャを実行して、各オブザベーションのサイズを表示します。最後に、データセットのオブザベーションの数に対してサイズを乗算し、処理が必要なバイト数の合計を取得します。パフォーマンス統計量をより小さなデータセットの統計量と比較すると、大きなデータセットのパフォーマンスがサイズに比例しているかどうかを判断することができます。比例していない場合は、まだ最適化の余地があります。

注: この手法で得られるデータセットのサイズは、見積もりにすぎません。変数名の格納方法などの内部的な要件によって、実際のデータセットのサイズが多少異なる場合があります。

13 章

並列処理のサポート

概要

SAS では、複数の Base SAS プロシジャを導入するとともに SAS 9 より*スレッド*技術を導入し、(部分的に)複数の*スレッド*で実行するように拡張されました。SAS は、独自の内部サブシステムによって提供されるスレッド化処理機能を利用する製品およびコンポーネントの開発と拡張を継続しています。スレッド化は、複数 CPU が搭載されたローカルのデスクトップから、ハイパフォーマンスプラットフォームのサーバーまでさまざまなプラットフォームで使用可能です。これらのハイパフォーマンスサーバーには、通常は分散*クラスタ*として構成される、大容量マルチコアの*対称型マルチプロセッサ*(SMP)システムおよび*超並列処理*(MPP)*アプライアンス*が含まれます。これらのプラットフォームで実行する多くの SAS コンポーネントでスレッド技術を利用しています。

Base SAS 9.4 では、SAS DS2 プログラミング言語または SAS 統合 SQL 言語を使用して作成されたプログラムで、スレッドを利用できます。これ以外の SAS 製品の多くもスレッド技術を使用しています。たとえば、SAS High-Performance Analytics プロシジャ、SAS Stored Processes、および SAS Embedded Process は、ハイパフォーマンスな分散コンピューティング環境で実行するコードを、実行あるいは生成します。

SAS のスレッド技術とは

スレッド技術は、動作環境に複数の実行パスを提供します。それぞれの実行パスはスレッドと呼ばれ、スレッドごとにプログラムタスクやデータ転送を扱えます。その結果は、並列で同時に実行される複数のプログラムタスクとデータ I/O 操作になります。スレッドには、コンテキスト(レジスタセットとプログラムカウンタなど)、実行するコードのセグメント、処理に使用する相当量のメモリが必要です。スレッド動作環境では、CPU が複数であっても、1CPU あたりの*コア*は 1 つだけです。さらにハイパフォーマンスな別の構成では、1CPU あたり複数のコアを持つ複数の CPU が含まれていて、1 コアあたり複数スレッドになる場合もあります。各 CPU が一度に 1 スレッドのみを実行する状況では、スレッド間の切り替えをすばやく行う CPU 処理能力によって、ほぼ同時の実行が実現されます。

SAS ソフトウェアのスレッド実行には、これら 2 つの一般的な技術の 1 つあるいは両方が含まれています。

- *スレッド I/O* とは、スレッド内でアプリケーションにデータ(たいていは非常に大容量)が提供されることを意味します。これにより、アプリケーションはデータを待つことなく、処理を継続できます。Base SAS および SAS/STAT では、スレッド化された読み込みをいくつかのプロシジャで利用しています。さらに、Base SAS には SPD Engine が含まれており、アプリケーションへのスレッド化された入力を最適化するため、区分データセットから読み込みを行います。SAS High-Performance Analytics プロシジャは、非常に高速のデータ提供を要求します。つまり、あるコンピューティングクラスタの全域に分散しているデータからスレッド化された読み込みを行い、巨大な量のデータをアプリケーションへ提供し(それもそのクラスタで処理されます)、そしてそのデータをデータストレージアプライアンスへ並列に書き込むことが要求されます。
- *スレッド化アプリケーションの処理* とは、アプリケーション自身が、複数 CPU マシンで特定タスクの並列処理が行える構造を持つことを意味します。スレッド化アプリケーションの処理では、小さいセグメントのデータを複数スレッドで実行するため、アプリケーションは大容量のデータをより高速に処理できます。ローカルの four-way デスクトップか、サーバークラスのマシンかにかかわらず、複数 CPU 搭載のマシンを利用するようにアプリケーションを設計できます。SAS High-Performance Analytics Server は、そのアプライアンスノード全域にデータとアプリケーションのコピーの両方を提供するアプライアンスで実行されるため、データはアプリケーション処理と同じ場所にあるといえます。

SAS 9.4 と SAS Analytics 12.1 では、スレッド化を使用して、計算が複雑なアルゴリズムやモデルだけでなく、常に増加するデータ量をサポートしている多種多様な製品やコンポーネントにアクセスできます。Base SAS および Foundation SAS のスレッド技術はこれらのすべてをサポートします。

SAS のスレッド制御方法

多くの SAS コンポーネントは、スレッド技術を自動的に利用しています。SAS 内部のメカニズムが、特定の環境変数を検出し、そのアプリケーションに適したパフォーマンスに依存してスレッド化を使用したり使用しなかったりできます。環境変数やシステムオプションには、管理者が構成可能なものがあります。使用可能な場合には、データセットオプションも、スレッドでの I/O やアプリケーション処理に影響を与える指定ができます。多くのコンポーネントではスレッドを自動的に使用しているので、スレッドを制御す

る特定のオプションがオフになったときでも、スレッドの使用はそのまま継続されると考えられます。

システムオプションの THREADS、NOTHREADS、CPUCOUNT は、スレッド化が自動ではない場合に、SAS 全体のスレッド処理に影響を与えます。SAS/ACCESS の DBSLICE などのように、スレッドの制御に追加オプションが必要な製品もあります。システムオプション THREADS は全製品でデフォルトのため、スレッド処理が使用可能でパフォーマンスが向上する場合は常にスレッド処理が行えます。NOTHREADS は、Base SAS または SAS クライアントにおいて、SMP(対称型マルチプロセッサ)環境で実行されている製品では、スレッド処理を使用不可にします。プロシジャステートメントオプションは、必要な場合に、システムオプションを上書きするために提供されます。

SAS/STAT、SAS/OR、SAS/ETS、SAS Enterprise Miner、SAS High-Performance Analytics Server プロシジャなど、SAS 製品の特定のプロシジャは、SAS クライアントの SMP モード、または*分散コンピューティング*環境の超並列処理モードで実行されます。SMP モードにおいては、NOTHREADS が設定されている場合はそちらが優先されます。THREADS が設定されている場合は、CPUCOUNT がデフォルトでクライアントでの処理に使用可能なスレッド数に設定されますが、調整は可能です。MPP モードでは、常にスレッドが想定されます。NOTHREADS は無視され、スレッド処理が常に有効です(クライアント SAS セッションまたは SAS Enterprise Miner から実行する場合を除く)。つまり、MPP モードでは NOTHREADS は有効ではありません。これらの製品のほとんどでは、スレッドの制御と実行モードは PERFORMANCE ステートメントで指定されます。その製品やコンポーネントで使用されるスレッド技術に関する情報については、特定の SAS 製品ドキュメントだけでなく、*SAS System Options: Reference* も参照してください。

Base SAS におけるスレッド化処理

Base SAS ではいくつかのスレッド化が自動的に行われます。さらに、THREADS オプションはすべての Base SAS コンポーネントのデフォルトで、スレッド化された読み込みやスレッド化されたアプリケーション処理をサポートしています。

基本言語
: SAS では、スレッド技術を SAS データファイルのインデックス構築に使用します。SAS 言語での WHERE 処理、BY グループ処理、SET ステートメントと MODIFY ステートメント、DO ループの ARRAY 処理には、インデックスによりパフォーマンス向上が期待できます。インデックスの構築に使用される、Base SAS の並べ替えアルゴリズムはデフォルトでスレッド対応になっていますが、NOTHREADS を使用して無効化できます。その製品やコンポーネントで使用されるスレッド技術に関する詳細については、特定の SAS 製品ドキュメントだけでなく、*SAS System Options: Reference* も参照してください。

スレッド対応 Base SAS プロシジャ
: 特定の Base SAS プロシジャには、スレッド化処理を利用できるアルゴリズムがあります。これらのプロシジャはスレッド対応で、アルゴリズムのいくつかの部分をスレッドで実行するように、プロシジャアルゴリズムの一部が分割されています。たとえば、SORT プロシジャはスレッド対応なので、並べ替えは使用可能なスレッド内で実行され、各スレッドでデータの一部が並べ替えられます。プロシジャは、複数スレッドから並べ替えられた順番にデータセットをすばやく生成します。これらのプロシジャはスレッドでデータを読み込むこともできます。

  そのプロシジャに対して使用可能なスレッドと CPU の数は、システムオプションまたはプロシジャオプションの CPUCOUNT および THREADS|NOTHREADS で指定します。NOTHREADS は、スレッド化処理をサポートしている SAS アプリケーションの実行時にスレッド化処理を使用しないことを指定します。THREADS がデ

フォルトです。NOTHREADS が有効な場合、CPUCOUNT は無視されます。Base SAS スレッド対応プロシジャは次のとおりです。

- MEANS
- REPORT
- SORT
- SUMMARY
- TABULATE
- SQL

詳細については、“Threaded Processing for Base SAS Procedures” (*Base SAS Procedures Guide*)を参照してください。スレッド対応 SQL プロシジャの詳細については、*SAS SQL Procedure User's Guide* を参照してください。. SAS システムオプションの詳細については、*SAS System Options: Reference* を参照してください。

SAS/STAT ソフトウェアのプロシジャにはスレッド対応のものもあり、それらのほとんどは SMP モードまたは MPP モードで実行できます。SMP モードの場合は、NOTHREADS および CPUCOUNT が優先されます。MPP モードの場合は、PERFORMANCE ステートメントがスレッド化を制御するオプションを提供します。スレッド対応 SAS/STAT プロシジャは次のとおりです。

- ADAPTIVEREG
- FMM
- GLM
- GLMSELECT
- LOESS
- MIXED
- QUANTLIFE
- QUANTREG
- QUANTSELECT
- ROBUSTREG

各プロシジャの詳細については、*SAS/STAT Procedures Guide* を参照してください。

SAS Scalable Performance Data (SPD) Engine
SAS Scalable Performance Data Engine (Base SAS に含まれています)は、SMP ハードウェアの性能を有効に引き出すように設計されています。SAS Scalable Performance Data Engine では、スレッドでのデータ読み込みのために最適化された区分データセットを使用します。区分のサイズは、SAS Scalable Performance Data Engine PARTSIZE オプションで構成できます。THREADNUM および SPDEMAXTHREADS で、最適なスレッド化された読み込みが行われるようにスレッドを制御します。Base SAS NOTHREADS システムオプションおよび CPUCOUNT システムオプションは、SPD Engine のスレッド化された読み込みでは有効ではありません。それらは、SPD Engine データセットに対して実行する SAS スレッド対応プロシジャには有効なままです。NOTHREADS が設定されていても、SPD Engine インデックスも自動的にスレッドで並列に作成されます。スレッド化されたインデックス作成の最適化には、SPDEINDEXSORTSIZE=が使用できます。SPD Engine については、*SAS Scalable Performance Data Engine: リファレンス*を参照してください。

SAS FedSQL 言語

SAS FedSQL は、ANSI SQL:1999 年標準に基づいた、SAS 独自の SQL 実装です。ANSI SQL データタイプおよびその他の ANSI 準拠機能をサポートしています。SAS FedSQL の主な利点として、異機種データベース環境で*統合クエリ*を実行し、単一の結果セットを返す機能が挙げられます。FedSQL クエリは、大規模操作をうまく処理するために複数スレッドアルゴリズムで自動的に最適化されます。また、FedSQL は、SAS セッションの外部、たとえば、SAS Federation Server や SAS Scalable Performance Data Server 環境内で実行できます。NOTHREADS オプションおよび CPUCOUNT オプションは FedSQL 処理では無効です。

FedSQL プログラムをサブミットして実行するための、FedSQL プロシジャが含まれます。完全な情報については、*SAS FedSQL Language Reference* を参照してください。

SAS DS2 プログラミング言語

DS2 は、SAS 独自のプログラミング言語で、高度なデータ操作やデータをモデリングするアプリケーションに適しています。DS2 は、Base SAS に含まれ、SAS DATA ステップと接点がありますが、追加のデータタイプ、ANSI SQL タイプ、プログラミング構造要素、ユーザー定義の方式、パッケージもサポートしています。DS2 SET ステートメントは、埋め込み FedSQL 構文に対応しているため、ランタイムで生成されたクエリにより、DS2 とサポートされている任意のデータベース間で対話的にデータを交換できます。これにより、2 つの言語の能力を効果的に組み合わせて入力テーブルの SQL 前処理ができます。

DS2 プログラムは、並列処理を実行するために作成されたプログラムで THREAD ステートメントを使用することで、スレッド対応になります。NOTHREADS オプションおよび CPUCOUNT オプションは無効です。ユーザーの DATA ステッププログラムを DS2 に変換することが有効かどうかの詳細については、SAS 9.4 DS2 Language Reference を参照してください。

SAS Embedded Process にサブミットしてスレッド対応 DS2 プログラムを実行する、DS2 プロシジャも含まれています。DS2 プロシジャのハイパフォーマンスバージョン、PROC HPDS2 では、DS2 言語ステートメントを処理するため、別々にライセンスされた High-Performance Analytics Server にサブミットします。このプロシジャやその他の特定の SAS プロシジャのハイパフォーマンスバージョンの詳細については、*SAS High-Performance Analytics Server Usage Guide* を参照してください。

DS2 は SAS セッションの外部で実行可能です。次にその例を示します。

- SAS Federation Server
- SAS Scalable Performance Data Server
- MPP コンピューティング環境(SAS In-Database Scoring Accelerator、SAS Embedded Process 環境、SAS High-Performance Analytics Server 分散環境など)

SAS ログ

SAS ログ機能は、NOTHREADS オプションと CPUCOUNT オプションを無視します。スレッドでのすべての入力ログイベントが扱われます。現在のスレッドやタスクに関係づけられているクライアント ID がログにレポートされます。ログ機能は、多くの SAS 製品やコンポーネントをサポートしていますが、Base SAS に含まれています。*SAS Logging: Configuration and Programming Reference* を参照してください。

SAS Code Analyzer

コメントを記録された SAS ジョブのメタデータを生成する場合、SAS Code Analyzer (SCAPROC プロシジャ)は、既存の SAS プログラムを(通常のプログラム実行で)実行します。PROC SCAPROC は、実行する SAS ジョブから、ジョブステップに関する情報、ファイルの依存関係などの I/O 情報、マクロシンボルの使用情報を取得します。出力はコメントを含む SAS プログラムで、そのコメントに依存関係が記述さ

れています。アプリケーションは、このテキストを読み込んで SAS メタデータを作成するか、これらの依存関係に基づいてプロセスフローを決定できます。たとえば、SAS Data Integration Studio の開発では、SAS Code Analyzer から出された情報を使用して以前の SAS ジョブをリバースエンジニアリングできます。これは、SAS Grid Manager でも使用できます。保存されたジョブを*グリッド*で実行する場合、SAS Grid Manager は識別されたサブタスクを自動的にグリッドノードに割り当てます。詳細については、*Base SAS Procedures Guide* で SCAPROC プロシジャの説明を参照してください。

SAS/ACCESS エンジン

SAS/ACCESS エンジンは、60 を超えるリレーショナルデータベースや非リレーショナルデータベース、PC ファイル、データウェアハウスアプライアンス、分散ファイルシステムに、読み込み、書き込み、および更新アクセスを提供する *LIBNAME エンジン*です。これらのエンジンは、Base SAS の一部ではありませんが、Base SAS を基にしています。これらは、別々にライセンスされているか、SAS BI Server または SAS Activity-Based Management などの多くの製品に同梱されています。同梱されている多くの製品では、使用可能な多数の SAS/ACCESS エンジンから 2 つの選択ができます。

SAS/ACCESS エンジンを使用して、SAS プログラムは、SAS データセットであるかのように、DBMS に接続できます。これは、バルクロードのサポート、一時テーブルのサポート、明示的なパススルーを使用するネイティブ SQL サポートを含む、パフォーマンスに関連する DBMS 機能と利点を利用します。DBMS が並列サーバーの場合、エンジンは複数スレッドを使用して DBMS サーバーに接続し、DBMS データに並列アクセスします。SAS プログラムがスレッド対応 SAS プロシジャを SAS/ACCESS エンジンを使用して実行している場合、パフォーマンスに大きな向上が見込まれます。

SAS/ACCESS でのスレッド化された読み込みでは、結果セットは複数スレッドにわたって分割されます。Base SAS プロシジャでのスレッド化処理とは異なり、SAS/ACCESS でのスレッド化された読み込みは、マシンのプロセッサの数に依存しません。そのかわりに、結果セットは SAS と DBMS 間の複数接続で取得されます。SAS で WHERE 句を SQL ステートメントに追加すると、DBMS は結果セットを分割します。これにより、単一 SQL ステートメントは複数 SQL ステートメントとなり、各スレッドに 1 つのステートメントということになります。また、DBMS は、1 スレッドあたり 1 つの区分を読み込みます。

SAS/ACCESS エンジンでの*スケーラビリティ*のレベルは、DBMS 自身に実装された並列化の処理効率に依存します。ただし、SAS/ACCESS エンジンは、LIBNAME ステートメントのオプションを使用して、SAS/ACCESS エンジン内でスレッドの実装の調整ができます。SAS/ACCESS でスレッド化された読み込みを制御するオプションは、DBSLICE、DBSLICEPARM、THREADS|NOTHREADS です。また、BY、OBS または KEY オプションが PROC または DATA ステップで使用されるかどうかも制御されます。詳細については、SAS/ACCESS for Relational Databases ドキュメントを参照してください。

SAS Scalable Performance Data Server

SAS Scalable Performance Data Server は、複数ユーザー並列処理データサーバーで、総合的なセキュリティ基盤、バックアップと復元ユーティリティ、および管理と調整のオプションを備えています。SPD Server はネイティブ SQL、FedSQL、および DS2 言語をサポートします。SPD Server に特化したオプションで、スレッド処理を制御します。NOTHREADS オプションおよび CPUCOUNT オプションは無効です。詳細について

は、*SAS Scalable Performance Data Server:Administrator's Guide* および *SAS Scalable Performance Data Server:User's Guide* を参照してください。

SAS Intelligence Platform

SAS Intelligence Platform は、企業のインテリジェンスを作成、管理、配布するインフラストラクチャです。このインフラストラクチャは、金融サービス、ライフサイエンス、ヘルスケア、小売業、製造業などの産業界の SAS ソリューションをサポートしています。SAS Intelligence Platform は、Base SAS の一部ではありませんが、Base SAS および次のコンポーネントを含む SAS Foundation に依存します。

- メタデータ定義のための SAS 管理コンソール
- SAS Business Intelligence Server および SAS Data Integration Server 技術
- 次のものを提供する、SAS Integration Technologies
  - SAS サーバーおよびサポートしているサービス(SAS Stored Process Server など)
  - アプリケーションメッセージング
  - SAS BI Web サービス
  - フレームワークのパブリッシュ
  - SAS Foundation Services

SAS Intelligence Platform Overview には、個々のコンポーネントが説明されており、広範囲の SAS Intelligence Platform 関連ドキュメントを参照できます。

Intelligence Platform の SAS サーバーは次のとおりです。

- SAS Workspace Server
- SAS Stored Process Server
- SAS Pooled Workspace Server
- SAS OLAP Server
- SAS Metadata Server

各サーバーはアクティブなスレッドのプールを伴って開始されます。これらのスレッドはサーバーが制御し、サーバープロセスによって使用されます(たとえば、入力要求の取り扱い)。NOTHREADS オプションと CPUCOUNT オプションが指定されているとき、これらのオプションを優先する SAS プロシジャを含むコードをサブミットして実行している場合を除き、これらのオプションは無視されます。

SAS Metadata Server の場合、スレッドの使用は、オブジェクトサーバーパラメータ(THREADSMAX および THREADSMIN)のデフォルト設定と、メタデータサーバー構成オプション、MACACTIVETHREADS のデフォルト設定で制御されます。管理者はこれらの設定を上書きして、パフォーマンスを微調整できます。詳細と例については、*SAS Intelligence Platform:System Administrator's Guide* を参照してください。THREADMAX および THREADMIN オブジェクトサーバーパラメータは、SAS Metadata Server 以外のサーバー用にはほとんど使用されません。

Intelligent Platform 中間層(Web アプリケーションのインフラストラクチャ)では、入力要求はスレッド上で処理されます。これらのスレッドが、ジョブの実行サービスを使用して定義されます。それらのスレッドは NOTHREADS オプションや CPUCOUNT オプションには制約を受けません。ジョブキュースレッドの数とジョブ実行スレッドの数の両方

を指定できます。*SAS Intelligence Platform:Middle-Tier Administration Guide* を参照してください。

SAS MP CONNECT は、Intelligence Platform に同梱される SAS/CONNECT ソフトウェアの一部分です。複数 SAS セッション間に 1 つの接続を確立し、それぞれのセッションが非同期にタスクを並列実行できるようにすることで、並列処理をサポートしています。同じローカルコンピュータの複数プロセスへ接続を複数確立することにより、アプリケーションはネットワークリソースを使用して並列処理を行い、すべての結果をクライアント SAS セッションにまとめることができます。多くの SAS プロセスは、SMP コンピュータではマルチプロセッサを使用しますが、ネットワーク上では、リモートにある複数のシングルプロセッサまたはマルチプロセッサコンピュータで実行できます。スレッドは常に使用可能であると想定されています。

MPP 環境として構成されている場合に SAS Intelligence Platform で実行可能な SAS High-Performance Analytics 製品があります。たとえば、SAS Grid Manager(次のセクションで説明します)は、SMP 構成で実行する SAS アプリケーションのワークロード管理を扱います。さらに、この製品は、並列実行用に作成され、SAS High-Performance Analytics Server のノード全体にわたって配布されるアプリケーションも管理できます。

SAS Visual Analytics は、MPP 環境として構成されている場合は SAS Intelligence Platform で実行可能な、ハイパフォーマンス分析アプリケーションの Web ベーススイートです。この SAS Visual Analytics の実行環境については、*SAS Intelligence Platform:Middle-Tier Administration Guide* を参照してください。(SAS Visual Analytics については次のセクションで詳しく説明します。"SAS High-Performance Analytics ポートフォリオの製品" (202 ページ)を参照してください。)

SAS High-Performance Analytics ポートフォリオの製品

SAS High-Performance Analytics 製品はスレッドを高度に利用するために設計されています。これらの製品は Base SAS や SAS Foundation の一部ではありません。しかしながら、これらの製品は、Base SAS 機能に依存しそれを拡張して、巨大なデータ量の高速分析をサポートする機能を提供します。

SAS High-Performance Analytics 製品には、その他の SAS アプリケーションやソリューションと協調したり、直接的に統合したりして動作するものがあります。たとえば、SAS Grid Manager を構成して、SAS Intelligence Platform で実行される SAS Enterprise Guide のワークロードを分散できます。SAS Grid Manager は、EMC Greenplum や Teradata などの DBMS アプライアンスで実行されている SAS High-Performance Analytic Server 上の SAS ジョブのワークロードを管理するために使用できます。SAS Visual Analytics は、SAS グリッド環境で実行されている SAS Enterprise Miner が使用するデータの探索に使用できます。

SAS High-Performance Analytics 技術には次のものが含まれます。

- SAS Grid Manager
- SAS In-Database
- SAS In-Memory Analytics 技術。これには次のものがあります。
  - SAS High-Performance Analytics Server
  - SAS Visual Analytics
  - SAS High-Performance Risk Management
  - その他の SAS High-Performance 製品およびソリューション

これらの技術のより完全な情報については、SAS High-Performance Analytics Web サイトを参照してください。Products & Solutions/High-Performance Analytics。

SAS Grid Manager

SAS Grid Manager は、構成されたサーバーのグリッド上のスレッドで実行されるさまざまな SAS プロセスとサービスにわたり、スケーラブルなワークロード管理と優先順位付け、高可用性処理、パフォーマンスの最適化を提供します。別々のサーバーにあるリソースを中央管理された SAS 環境に統合することで、SAS Grid Manager が SAS ジョブの処理を加速できます。

プログラムフロー全体が複数の独立したステップで実行される SAS プログラムはグリッド対応であり、その SAS プログラムはスレッドで実行されます。スレッド化された実行は一般的に DATA ステップまたはプロシジャ境界で行われます。プログラムはスレッドで実行されることが明確に書かれるため、THREADS オプションは設定済みで CPUCOUNT オプションは 1 より大きいとみなされます。SAS Grid Manager を使用すると、独立した SAS ジョブのサブタスクがグリッドの異なる部分で並列に実行されてリソースのプールを共有できます。スレッド化されたサブタスクは、コード内から任意のスレッド対応プロシジャが実行され、それが実行するスレッド化処理で効果を得られます。SAS Code Analyzer を使用して並列処理用に分析されたプログラムは、SAS Grid Manager を使用するグリッドで実行できます。その場合、識別されたサブタスクは自動的にグリッドノードに割り当てられます。

SAS Enterprise Guide、SAS Enterprise Miner、SAS Data Integration Studio、SAS Web Report Studio、SAS Marketing Automation、SAS Marketing Optimization などの多くの SAS 製品は、SAS Grid Manager と統合されています。アプリケーション内、または SAS Metadata 内で使用されるオプションが、統合を可能にします。SAS Data Integration Studio および SAS Enterprise Miner には、コード生成エンジンがあります。このコード生成エンジンが並列処理の機会を認識し、SAS Grid Manager にサブミットしてグリッド全体で並列実行する適切なコードを生成します。

SAS Intelligence Platform などのその他の SAS コンポーネントでは、SAS Grid Manager を使用して、ネットワーク上の複数コンピュータ全体でサーバーワークロードを分散することによる処理に最適な SAS サーバーを判別します。SAS Grid Manager はジョブを別々のプロセスに分割し、複数のサーバー全体で並列に実行させます。SAS Grid Manager については、*Grid Computing for SAS* を参照してください。

SAS In-Database 技術

SAS In-Database 技術は、SAS アプリケーションがコマンドや関数をデータベースの内部で実行するように送り込むことで、実行を高速化します。このプロセスでは、データの移動と変換をしなくてよく、ホストデータベースのスレッド化機能も利用できます。

SAS In-Database は、Teradata、Greenplum、Aster Data、Netezza、DB2 などの、スレッド化されたさまざまな DBMS とハードウェアまたはソフトウェアアプライアンスアーキテクチャ上で実行されます。スレッド化はデータベースが制御するため、NOTHREADS オプションおよび CPUCOUNT オプションは無効です。

In-Database 処理は、データベース内で複数の並列タスクに分割され、処理されるデータ全体の小さなサブセット同士で連携します。サブセットの結果が出されると、それらは 1 つにまとめられて SAS アプリケーションに戻されます。一般的に、実行時間は、データが SAS サーバーに転送された場合よりはるかに短くなります。

In-Database 技術は、データベースの既存機能と SAS 標準機能を使用して、SAS 機能をデータベースに拡張しています。たとえば、SORT または SUMMARY などの Base SAS プロシジャを、プロシジャのコードに対する単純なオプションを含んだデータベース環境内で実行できます。これらのプロシジャは、データベース環境内で実行するために、ネイティブのデータベース関数に“翻訳”されます。SAS 機能のデータベースへの拡張例としては、SAS Enterprise Miner が生成したスコアリングコードが関数としてエクスポートできることがあります。エクスポートした後で、これらの関数は、SAS プロセスを使用するか、またはサードパーティやデータベース固有のアプリケーションから標準 SQL ステートメントを使用して呼び出すことで実行できます。

SAS In-Database 技術については、*SAS In-Database Products:Administrators Guide* と *SAS In-Database Products:User's Guide* を参照してください。SAS In-Database コンポーネントには、サポートされるほとんどのデータベースのスコアリングアクセラレータと分析アクセラレータが含まれます。

SAS In-Memory Analytics 技術

SAS In-Memory Analytics 技術では、多数のスレッドや高水準のメモリを利用します。これらは Teradata や EMC Greenplum などの特別に構成されたいくつかの DBMS アプライアンスや、*Hadoop 分散ファイルシステム*(HDFS)を使用したコモディティハードウェアで使用可能です。Teradata と EMC Greenplum アプライアンスは、特定の MPP (超並列処理)技術を使用するコンピューティングクラスタとして組み立てられています。SAS In-Memory Analytics 技術では、処理されるデータのすべてがクラスタ全体に分散され、分析プロシジャが開始する前にメモリにロードされます。これは、データが必要なときにブロックでロードされる従来の処理とは明確に異なる点です。さらに、SAS High-Performance Analytics プロシジャは、数百のスレッドで実行されるように設計され、各スレッドは処理されるデータ全体の小さなサブセットに対する役割を担っています。大きなデータセットを速く分析できるようになることで、分析モデルがより洗練されます。

SAS High-Performance Analytics ファミリ製品のメンバのいくつかは、SAS High-Performance Analytics Server、SAS Visual Analytics、SAS High-Performance Risk を含む、SAS In-Memory Analytics 技術に基づいています。多様な SAS In-Memory Analytics コンポーネントの詳細については、次の In-Memory Analytics Web サイトを参照してください。Products & Solutions/In-Memory Analytics.

SAS High-Performance Analytics Server

SAS High-Performance Analytics Server は、複数スレッドで実行するよう設計され、複雑な計算を伴う予測モデルの開発に焦点を合わせたハイパフォーマンス分析プロシジャを提供します。これらのプロシジャは、SAS/STAT、SAS/QC および SAS/ETS のライブラリから呼び出されます。プロシジャは、EMC Greenplum や Teradata アプライアンス、および Hadoop クラスタで提供される MPP コンピューティング環境で実行します。ハイパフォーマンス MPP 構成には一般的に、最小 1.5TB のメモリと、1 コアごとに複数スレッドを処理する 192 以上のコアがあります。

SAS High-Performance Analytics プロシジャは、Base SAS セッションが実行されている、要求側 SAS クライアントで起動します。起動元は、従来の Display Manager System、SAS Enterprise Guide、または SAS Enterprise Miner の SAS High-Performance データマイニングタブのいずれかです。MPP 環境では、SAS クライアントが SAS High-Performance Analytics Server ノードと通信し、そのノードでシン SAS 環境が要求された SAS プロシジャまたは DS2 コードのコピーを実行します。分析結果は完了後に SAS クライアントの要求側アプリケーションに戻されます。

SAS High-Performance Analytics プロシジャの PERFORMANCE ステートメントを使用して、スレッド化や処理モード、SMP (クライアントモード)または MPP (分散モード)を制御するパラメータが指定できます。SMP モード(クライアントマシンでの

SAS セッション)では、CPUCOUNT のデフォルトはクライアントマシンの CPU 数です。CPUCOUNT オプションと NOTHREADS オプションは SAS システムオプションを上書きできます。この環境では、プロシジャが MPP モードで実行される場合、CPUCOUNT オプションのデフォルトは、アプライアンスノードの CPU 数でスレッドの数が決定されることです。PERFORMANCE ステートメントでのみ使用可能な NTHREADS オプションは、スレッドの数を調整します。

SAS High-Performance Analytics Server およびプロシジャについては、*SAS High-Performance Server Administration Guide* を参照してください。SAS High-Performance Analytics Server は、SAS LASR Analytic Server に依存し、大量データおよび複雑な計算のために最適化された、高度にスケーラブルで信頼できる分析のインフラストラクチャを提供します。

SAS Visual Analytics

MPP 環境として構成される場合、SAS Visual Analytics は SAS High-Performance Analytics Server および SAS Intelligence Platform 上で稼働します。SAS Visual Analytics は、ビジネスユーザーおよびビジネスアナリストに、大きなデータを探査し値を引き出すために特別に設計された視覚化表現を使用して多種多様のタスクを実行する機能を提供します。これにより、記述統計量を超え、より専門的な分析を非常にスケーラブルな方法で使用できます。将来へのより正確な洞察力を提供し、意思決定を支援します。たとえば、ユーザーは視覚的にデータを探索して、何十億行のデータの相関分析を数秒で実行し、視覚的に結果を示すことができます。以前は明白でなかったデータのパターン、傾向および関係の迅速な識別に役立ちます。

SAS Visual Analytics と、SAS High-Performance Analytics Server および SAS In-Database を組み合わせることで、ハイパフォーマンスモデルのライフサイクルを実現できます。たとえば、SAS Visual Analytics を使用して、データを探索してどの情報を使用するかを決定します。SAS High-Performance Analytics Server を使用してユーザーモデルを作成し、SAS In-Database を使用してそのモデルを適切なデータベースに格納してスコアリングできるようにします。

SAS Visual Analytics には、Teradata や EMC Greenplum アプライアンス、または MPP クラスタとして構成された Hadoop HDFS などのブレードハードウェアの専用で特殊な構成が必要です。この環境は常にスレッド処理されます。SAS オプション CPUCOUNT および NOTHREADS は無効です。かわりに、PERFORMANCE ステートメントの NTHREADS オプションが、スレッドの使用を調整する方法を提供します。製品の情報については、*the SAS Visual Analytics:User's Guide* を参照してください。

SAS Visual Analytics は SAS LASR Analytic Server に依存し、大量データおよび複雑な計算のために最適化された、高度にスケーラブルな分析のインフラストラクチャを提供します。

SAS LASR Analytic Server

SAS LASR Analytic Server は、データへの同時アクセスに安全な環境を提供する分析プラットフォームです。SAS High-Performance Analytics Server のコンピューティングノード全体にわたって、データをメモリにロードします。SAS LASR Analytic Server は、SAS High-Performance Analytics Server ルートノードで、アプライアンス全体の並列処理でデータをメモリに高速で読み込むワーカーノードを使用して実行されます。データが*共存データプロバイダ*から提供されない場合、DBMS アプライアンスまたは Hadoop クラスタからデータが読み込まれ、SAS High-Performance Analytics Server のルートノードへ転送されます。その後、データは*ワーカーノード*のメモリにロードされます。SAS LASR Analytic Server は、CPUCOUNT や NOTHREADS の影響を受けません。かわりに、PERFORMANCE ステートメントの NTHREADS オプションがスレッドの使用を調整します。詳細については、*SAS LASR Analytic Server:Administration Guide* を参照してください。

SAS In-Memory Analytics 製品の詳細については、Products & Solutions/In-Memory Analytics を参照してください。

SAS High-Performance Analytics 製品の統合

SAS High-Performance Analytics ポートフォリオは、多数の SAS ソリューションと製品をサポートしています。一部の SAS 製品は、*並列処理*を利用するために、SAS Grid Manager および(ある程度は)他のハイパフォーマンス分析製品と統合されています。

SAS Enterprise Guide:

SAS Enterprise Guide は、SAS クライアントとして実行され、SAS サーバーへのフロントエンドを提供し、SAS プログラムを実行します。Base SAS で並列処理機能を利用するプログラム(たとえば、スレッド対応プロシジャや Stored Processes)をユーザーが作成します。SAS Enterprise Guide は、SAS Grid Manager が環境を管理していて、ワークロードバランシング、リソース割り当て、およびジョブの優先順位付けを提供するかどうかを検出できます。SAS Enterprise Guide は、別のグリッドノードでプロセスフローブランチを並列に実行できます。これにより、同一サーバーでのタスクの並列実行が可能になり、SAS Grid Manager 環境でのプロジェクトや個人レベルでタスクを実行できます。詳細については、*SAS Enterprise Guide* を参照してください。

SAS Data Integration Studio

SAS Data Integration Studio は SAS Data Integration Server 上で稼働し、SAS Grid Manager がインストールされている場合にはグリッドコンピューティングを自動的に利用します。SAS Data Integration Studio 3.4 の機能拡張により、SAS Grid Manager が管理するグリッド上で実行できる SAS アプリケーションが自動的に生成されるようになりました。ユーザーは、プログラミングの知識やグリッドインフラストラクチャの基本知識がなくても、グリッド対応 SAS アプリケーションを作成できます。

これらの SAS アプリケーションは SAS Grid Manager 環境の存在を実行時に検出し、状況に応じて実行を分散します。これらのグリッド対応アプリケーションは SAS Stored Processes として保存され、後で SAS Web Report Studio、SAS Information Map Studio、SAS Add-In for Microsoft Office などの SAS Intelligence Platform コンポーネントによって実行できます。(SAS Foundation によっては、これらのアプリケーションには SAS BI または SAS Enterprise BI Sevrer が必要です)。

SAS Data Integration Studio は、SAS Intelligence Platform を使用して MPP 環境(常にスレッド化)として構成されたインメモリ製品である SAS High-Performance Analytics Server や SAS Visual Analytics Server とともに実行できます。SAS Data Integration Studio は、SAS LASR Analytic Servers または HDFS に、ハイパフォーマンス分析変換を提供します。NOTHREADS オプションおよび CPUCOUNT オプションは無効です。詳細については、*SAS Data Integration Studio: User's Guide* を参照してください。SAS Data Integration Server は、SAS Intelligence Platform の一部として管理されます。

SAS Risk Dimensions

SAS Risk Dimensions の反復ワークフローは SAS Data Integration Studio のものと同じです。どちらも、データの別サブセット上で同じ分析を実行します。このワークフローは、SAS Grid Manager を理想的に利用して、グリッド全体に処理を分散します。詳細については、*SAS Risk Dimensions User's Guide* を参照してください。

SAS Enterprise Miner

SAS Enterprise Miner は、SAS Grid Manager グリッドで実行可能な SAS アプリケーションを自動的に生成できます。これらの SAS アプリケーションは、SAS Grid Manager 環境の存在を実行時に検出し、状況に応じて実行を分散します。アプリ

ケーションは、SAS Stored Processes としても保存可能で、保存後は SAS Web Report Studio などの SAS Business Intelligence コンポーネントによって実行することもできます。DMINE、DMREG および DMDB プロシジャは、デフォルトで THREADS が使用され、スレッド対応です。NOTHREADS は、複数スレッド演算を無効化します。

SAS Enterprise Miner は、データのビン化、補完、スコアリング、変換などのタスクのハイパフォーマンス統計やデータマイニングプロシジャのキーセットを含んでいます。これらのプロシジャは、高度にスレッド化された SAS High-Performance Analytics Server で実行されます。MPP モードでは、予測モデルを作成するサーバーノード全体に、SAS Enterprise Miner がデータ、メモリ、演算を分配します。SAS High-Performance Analytics Server がインストールされており、特定の Hadoop HPDM コードが有効化されている場合、SAS Enterprise Miner の HPDM タブが使用可能になりサーバーノードでの実行が許可されます。Enterprise Miner のスコアリングコードは、SAS In-Database 技術を使用してデータベース内部で実行できます。詳細については、*SAS Enterprise Miner:Administration and Configuration* を参照してください。

14 章

SAS レジストリ

SAS レジストリの概要

SAS レジストリについて

SAS レジストリは SAS の構成データのための中央記憶領域です。たとえば、レジストリは次の内容を格納します。

- SAS が起動時に割り当てるライブラリとファイルショートカット
- エクスプローラポップアップメニューのメニュー定義
- 使用するために定義したプリンタ
- さまざまな SAS 製品の構成データ

この構成データは階層形式で格納されます。この形式は、UNIX や Windows のような動作環境や、z/OS UNIX System Services (USS)のディレクトリベースのファイル構造と同様の方法で機能します。

注: ホストプリンタは SAS レジストリで参照されます。

SAS レジストリの使用者について

SAS レジストリは、システム管理者や経験豊富な SAS ユーザーが使用するためのものです。このセクションでは、レジストリツールの概要を示し、レジストリの一部をインポートおよびエクスポートする方法について説明します。

注意:
レジストリの編集時に誤りがあると、お使いのシステムが不安定になったり、使用できなくなったりする場合があります。

可能であれば、管理ツールを使用します。たとえば、**ライブラリの作成**ウィンドウ、PRTDEF プロシジャ、ユニバーサル印刷ウィンドウ、**エクスプローラオプション**ウィンドウなどで、レジストリを直接編集するのではなく構成を変更できます。管理ツールを使用すると、構成を変更した場合にレジストリに値が正しく格納されます。

注意:
レジストリエディタを使用して値を変更する場合、エントリが不正であっても警告は出されません。エントリが不正であると、エラーが発生し、SAS セッションを起動できなくなることさえあります。

SAS レジストリの格納場所

Sasuser ライブラリと Sashelp ライブラリ内のレジストリファイル

SAS レジストリは、論理的には 1 つのデータストアですが、物理的には Sasuser ライブラリと Sashelp ライブラリ両方にある 2 つの異なるファイルで構成されます。レジストリの物理ファイル名は regstry.sas7bitm です。デフォルトでは、これらのレジストリファイルは、Sashelp ライブラリと Sasuser ライブラリの SAS エクスプローラビューでは表示されません。

- Sashelp ライブラリレジストリファイルには、サイトのデフォルトが含まれます。システム管理者は通常、サイトが使用するプリンタ、起動時に割り当てられるグローバルファイルショートカットやライブラリ、それ以外のサイトの構成デフォルトを構成します。
- Sasuser ライブラリレジストリファイルには、ユーザーのデフォルトが含まれます。**印刷設定**ウィンドウや**エクスプローラオプション**ウィンドウなど、特殊なウィンドウを介して構成情報を変更すると、設定は Sasuser ライブラリに格納されます。

サイトのデフォルトの復元法

オリジナルのサイトのデフォルトを現在の SAS セッションで復元する場合は、regstry.sas7bitm ファイルを Sasuser ライブラリから削除し、SAS セッションを再起動します。

SAS レジストリの表示方法

SAS レジストリを表示する場合、次の 3 つの方法のいずれかを使用できます。

- **REGEDIT** コマンドを発行します。SAS レジストリエディタが表示されます。
- **ソリューション** ⇨ **アクセサリ** ⇨ **レジストリエディタ**を選択します。
- コードの次の行をサブミットします。

```
proc registry list;
run;
```

この方法では、SAS ログにレジストリを出力し、サブキーを含め、レジストリエントリをすべて含む大きなリストを生成します。サイズが大きいため、この方法ではレジストリの表示に時間がかかることもあります。

SAS レジストリの表示方法については、“REGISTRY” (*Base SAS Procedures Guide*)の REGISTRY プロシジャを参照してください。*Base SAS Procedures Guide*。

SAS レジストリの定義

SAS レジストリは、DOS または UNIX のファイルシステムなどのディレクトリとサブディレクトリを使用するかわりに、キーとサブキーを構造の基本として使用します。これらの用語およびその他の用語については、次で説明します。いずれも、SAS レジストリを説明する際に頻繁に使用されます。

キー
SAS の特定の側面を参照するレジストリファイルのエントリです。レジストリファイルの各エントリは、キー名に続き、改行して 1 つ以上の値で構成されます。キー名は、1 行に入力し、角かっこ([および])で囲みます。

キーは、値や値に関連するサブキーのないプレースホルダでも、関連する値を持つ多くのサブキーがあってもかまいません。サブキーは、バックスラッシュ(\)で区切られます。単独のキー名またはキー名のシーケンスの長さは(角かっことバックスラッシュを含め) 255 文字を超えないようにしてください。キー名には、バックスラッシュを除く任意の文字を使用できます。また、大文字と小文字は区別されません。

SAS レジストリには、SAS_REGISTRY という最上位のキーのみが含まれています。SAS_REGISTRY の下のすべてのキーがサブキーです。

サブキー
別のキーの中のキーです。サブキーは、バックスラッシュ(\)で区切られます。サブキー名では、大文字と小文字を区別しません。次のキーには、1 つのルートキーと 2 つのサブキーが含まれています。**[SAS_REGISTRY\HKEY_USER_ROOT\CORE]**

SAS_REGISTRY
ルートキーです。

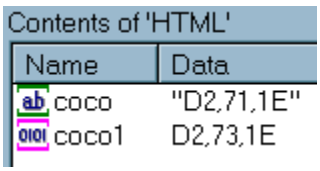
HKEY_USER_ROOT
SAS_REGISTRY のサブキーです。SAS レジストリでは、このレベルに別のサブキー(HKEY_SYSTEM_ROOT)があります。

CORE
プリンタやウィンドウなどのさまざまなデフォルト属性を含んでいる HKEY_USER_ROOT のサブキーです。

リンク
キーを参照する内容を含んでいる値です。リンクは、内部でのみ使用されます。リンクの値は必ず“link:”で始まります。

値
キーまたはサブキーに関連付けられた名前と内容です。値、値の名前、値の内容(別名: 値データ)に対して、2 つのコンポーネントがあります。

図 14.1 *サブキー'HTML'の値の名前と値のデータを表示するレジストリエディタのセクション*



.SASXREG ファイル
ファイル拡張子が.SASXREG のテキストファイルです。実際のバイナリ SAS レジストリファイルのテキスト表現を格納しています。

SAS レジストリの管理

SAS レジストリの管理に関する主な留意点

注意:
レジストリの編集時に誤りがあると、お使いのシステムが不安定になったり、使用できなくなったりする場合があります。 可能な場合は管理ツールを使用します。たとえば、**ライブラリの作成**ウィンドウ、PRTDEF プロシジャ、**ユニバーサル印刷**ウィンドウ、**エクスプローラオプショ**ウィンドウなどを使用すると、レジストリを直接編集せずに設定を変更できます。これは、構成を変更したときにレジストリに値を正しく格納するためのものです。

注意:
レジストリエディタを使用して値を変更する場合、エントリが不正であっても警告は出されません。 エントリが不正であると、エラーが発生し、SAS セッションを起動できなくなることさえあります。

Sasuser レジストリのバックアップ

Sasuser レジストリをバックアップする理由
レジストリの Sasuser[1] 部分には、ユーザーの個人的な設定が格納されています。SAS セッションを大規模にカスタマイズする場合は、レジストリの Sasuser 部分をバックアップすることを推奨します。大規模なカスタマイズには、次のものが含まれます。

- 新しいプリンタのインストール
- システム管理者がユーザー向けに行ったデフォルトプリンタ設定の変更
- ローカライゼーション設定の変更
- TRANTAB による変換テーブルの変更

デフォルト設定にリセットした場合
SAS の起動時に、レジストリファイルが自動的にスキャンされます。レジストリは、次の 2 つの条件に合致すると元の設定に復元されます。

- レジストリが壊れていることを検出した場合、ファイルが再構築されます。
- Sasuser ライブラリにある regstry.sas7bitm というレジストリファイルを削除した場合、Sasuser レジストリはデフォルト設定に復元されます。

  注意:
  Sashelp にあるレジストリファイルを削除しないでください。SAS を起動できなくなります。

[1] レジストリの Sashelp 部分には、サイトの全ユーザーに共通の設定が格納されています。Sashelp には書き込み保護が設定されており、システム管理者のみが更新できます。

レジストリのバックアップ方法

レジストリをバックアップする方法には 2 種類あり、それぞれ異なる結果になります。

方法 1:regstry.sas7bitm という Sasuser レジストリファイルのコピーを保存する。

この方法では、コピーした時点のレジストリがそのままコピーされます。レジストリの破損したコピーを復元するためにレジストリのそのコピーを使用する必要がある場合、コピー後にレジストリに加えた変更は失われます。ただし、元のデフォルトレジストリを復元するよりも、このバックアップファイルを使用する方が便利です。

方法 2:レジストリエディタまたは PROC REGISTRY を使用して、Sasuser レジストリの変更された部分をバックアップする。

結果はレジストリの連結コピーです。これは、バックアップファイルから復元されます。REGISTRY プロシジャの EXPORT=ステートメント、またはレジストリエディタの**レジストリファイルのエクスポート**ユーティリティを使用してバックアップファイルを作成すると、変更されたレジストリの部分が保存されます。このバックアップファイルがレジストリに復元されると、次の方法で現在のレジストリに連結されます。

- バックアップ日以降に Sasuser レジストリに追加された新しいキー、サブキー、値はすべて完全に、新しいレジストリに保持されます。
- SAS のインストール後に変更され、バックアップ後に再び変更された既存のキー、サブキー、値は上書きされ、復元するとバックアップファイルの値に戻ります。
- 既存のキーまたはサブキー(または元のデフォルト値を保持している値)は、復元後にデフォルト値を持つことになります。

エクスプローラを用いた SAS レジストリのバックアップ

エクスプローラを使用して SAS レジストリをバックアップするには、次の操作を実行します。

1. **EXPLORER** コマンドを使用するか、または**表示** ⇨ **エクスプローラ**を選択して、SAS エクスプローラを起動します。
2. **ツール** ⇨ **オプション** ⇨ **エクスプローラ**を選択します。

   エクスプローラオプションウィンドウが表示されます。
3. **メンバ**タブを選択します。
4. **ITEMSTOR** を**種類**リストで選択します。
5. **表示**をクリックします。

   ITEMSTOR が**種類**リストに関連アイコンを持っていない場合、アイコンを選択するように要求するメッセージが表示されます。
6. Sasuser ライブラリを**エクスプローラ**ウィンドウで開きます。
7. **Regstry.Itemstor** ファイルを右クリックします。
8. ポップアップメニューから**コピー**を選択し、**Regstry** ファイルをコピーします。**Regstry_copy** というファイル名が付けられます。

   動作環境の情報

   バックアップ用にレジストリファイルのコピーを作成する場合は、動作環境のコピーコマンドを使用することもできます。SAS エクスプローラ以外で参照する場合、ファイル名は **regstry.sas7bitm** です。z/OS では、Sasuser ライブラリが HFS ディレクトリに割り当てられている場合を除き、動作環境のコピーコマンドでレジストリファイルをコピーすることはできません。

レジストリエディタを用いた SAS レジストリのバックアップ

通常は、バックアップからレジストリを復元する場合に備えて新しいキーまたは値が保持されるので、レジストリエディタを使用して SAS レジストリをバックアップする方法が推奨されます。

レジストリエディタを用いて SAS レジストリをバックアップするには、次の操作を実行します。

1. REGEDIT コマンドを使用してレジストリエディタを起動します。
2. レジストリウィンドウの左ペインで最上位のキーを選択します。
3. レジストリエディタから、**ファイル** ⇨ **レジストリファイルのエクスポート**を選択します。

   名前を付けて保存ウィンドウが表示されます。
4. レジストリバックアップファイルの名前をファイル名フィールドに入力します。(オペレーティングシステムに応じたファイル拡張子が適用されます)。
5. **保存**をクリックします。

これにより、レジストリバックアップファイルが Sasuser に保存されます。**名前を付けて保存**ウィンドウで別の場所を指定すると、レジストリバックアップファイルの場所を制御できます。

レジストリ障害からの回復

このセクションでは、バックアップファイルでレジストリを復元する手順を示し、破損したレジストリファイルを修復する方法を示します。

SAS エクスプローラまたはオペレーティングシステムのコピーコマンドを使用して作成したレジストリバックアップファイルをインストールするには、次の操作を実行します。

1. 破損したレジストリファイルの名前を変更します。
2. バックアップファイルの名前を *regstry.sas7bitm* (レジストリファイルの名前)に変更します。
3. 名前変更したレジストリファイルを、前回のレジストリファイルがある Sasuser の場所にコピーします。
4. SAS セッションをもう一度開始します。

レジストリエディタで作成されたレジストリバックアップファイルを復元するには、次の操作を実行します。

1. REGEDIT コマンドを使用してレジストリエディタを起動します。
2. **ファイル** ⇨ **レジストリファイルのインポート**を選択します。
3. 前回エクスポートしたレジストリファイルを選択します。
4. **開く**をクリックします。
5. SAS を再起動します。

PROC REGISTRY で作成されたレジストリバックアップファイルを復元するには、次の操作を実行します。

1. プログラムエディタを開き、次のプログラムをサブミットして、作成済みのレジストリファイルをインポートします。

```
proc registry import=<registry file specification>;
run;
```

これにより、レジストリファイルが Sasuser ライブラリにインポートされます。

2. 適切なファイル名が付いていない場合、エクスプローラを使用してレジストリファイル名を regstry.sas7bitm に変更します。
3. SAS を再起動します。

破損したレジストリを修復するには、次の操作を実行します。

1. 破損したレジストリファイルを“registry”以外の名前(*temp* など)に変更します。
2. SAS セッションを開始します。
3. *temp* レジストリの場所を示すライブラリを定義します。

```
libname here '.'
```

4. REGISTRY プロシジャを実行し、Sasuser レジストリを再定義します。

```
proc registry setsasuser="here.temp";
run;
```

5. REGEDIT コマンドを使用してレジストリエディタを起動します。**ソリューション** ⇨ **アクセサリ** ⇨ **レジストリエディタ** ⇨ **すべて表示する**を選択します。
6. HKEY_USER_ROOT キーの下にある破損フィールドを編集します。
7. SAS セッションを閉じ、変更したレジストリを元の名前に戻します。
8. 新しい SAS セッションを開き、変更によって問題が解決したかどうかを確認します。

SAS レジストリを用いた色の管理

色と SAS レジストリの概要

SAS レジストリには、ほとんどの Web ブラウザで共通の色名の RGB 値が含まれています。これらの色は、ODS と GRAPH 出力で使用できます。RGB 値は 3 色(赤、緑、青)で構成され、各要素の範囲は 00 - FF (0 - 255)です。

色のレジストリ値は COLORNAMES\HTML サブキーにあります。

レジストリエディタを用いた色の追加

独自の新しい色値を作成するには、COLORNAMES\HTML サブキーのレジストリに値を追加します。

1. REGEDIT コマンドを使用して SAS レジストリエディタを起動します。
2. **COLORNAMES\HTML** サブキーを選択します。
3. **編集** ⇨ **新規作成バイナリ値**を選択します。ポップアップメニューが表示されます。
4. **値の名前**フィールドに色の名前を入力し、**値のデータ**フィールドに RGB 値を入力します。
5. **OK** をクリックします。

プログラムを用いた色の追加

独自の新しい色値を作成するには、SAS コードを使用して、COLORNAMES\HTML サブキーのレジストリに値を追加します。

最も簡単な方法は、REGISTRY プロシジャで予期されているレイアウトでファイルに色の値を書き込み、REGISTRY プロシジャを使用してファイルをインポートすることです。次の例では、スペイン語の色の名前をレジストリに追加しています。

```
filename mycolors temp;
data _null_;
   file "mycolors";
   put "[colornames\html]";
   put ' "rojo"=hex:ff,00,00';
   put ' "verde"=hex:00,ff,00';
   put ' "azul"=hex:00,00,ff';
   put ' "blanco"=hex:ff,ff,ff';
   put ' "negro"=hex:00,00,00';
   put ' "anaranjado"=hex:ff,a5,00';
run;

proc registry import="mycolornames";
run;
```

これらの色をレジストリに追加すると、SAS で提供された色の名前を使用できる場所であればどこでもこの色の名前を使用できるようになります。たとえば、次のコードで示すように、GOPTIONS ステートメントで色の名前を使用できます。

```
goptions cback=anaranjado;
proc gtestit;
run;
```

レジストリエディタの使用

レジストリエディタの使用が適している場合

レジストリの内容を参照する最もよい方法はレジストリエディタを使用することです。レジストリエディタは REGISTRY プロシジャをグラフィックで表したものです。経験豊富な SAS ユーザーはレジストリエディタを使用して次のことを実行できます。

- レジストリの内容を表示します。レジストリには、キーおよびキーに格納された値が表示されます。
- キーおよびレジストリに格納された値を追加、変更、削除します。
- 任意のキーから、レジストリファイルをレジストリにインポートします。
- 任意のキーから、レジストリの内容をファイルにエクスポートします。
- レジストリファイルをアンインストールします。
- レジストリファイルを SAS レジストリと比較します。

SAS ウィンドウ環境の多くのウィンドウが、プリンタ設定や色のプリファレンスなどを変更したときにレジストリを更新します。これらのウィンドウは、適切な構文やセマンティックを使用してレジストリを更新するので、SAS を調整する場合にこうした代替的な方法を用いることが最も良い手段です。

レジストリエディタの開始

レジストリエディタを実行するには、SAS コマンドラインで **REGEDIT** コマンドを発行します。**ソリューション** ⇨ **アクセサリ** ⇨ **レジストリエディタ**を選択して、レジストリウィンドウを開くこともできます。

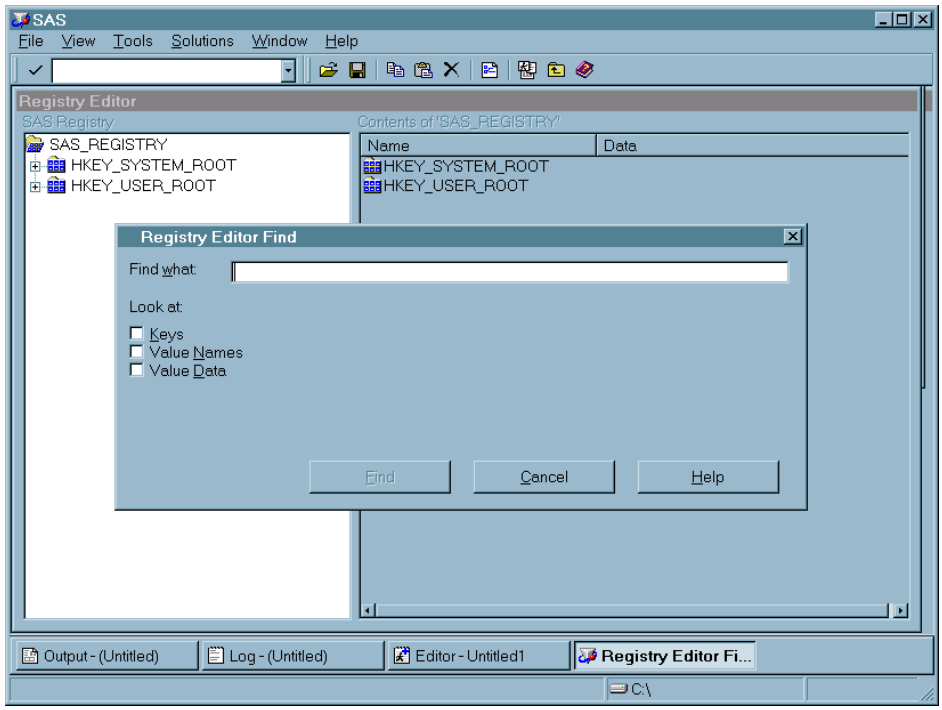
レジストリ内の特定のデータの検索

レジストリエディタウィンドウで、レジストリキーを格納しているフォルダアイコンをダブルクリックします。これにより、そのキーの内容が表示されます。

もう 1 つ、検索ユーティリティを使用する方法もあります。

1. レジストリエディタで、**編集** ⇨ **検索**を選択します。
2. 検索する文字列の全部または一部を入力し、**オプション**をクリックして、検索対象が**キー名**か、**値の名前**か、**データ**かを指定します。
3. **検索**をクリックします。

図 14.2 レジストリエディタの検索ユーティリティ



SAS レジストリ内の値の変更

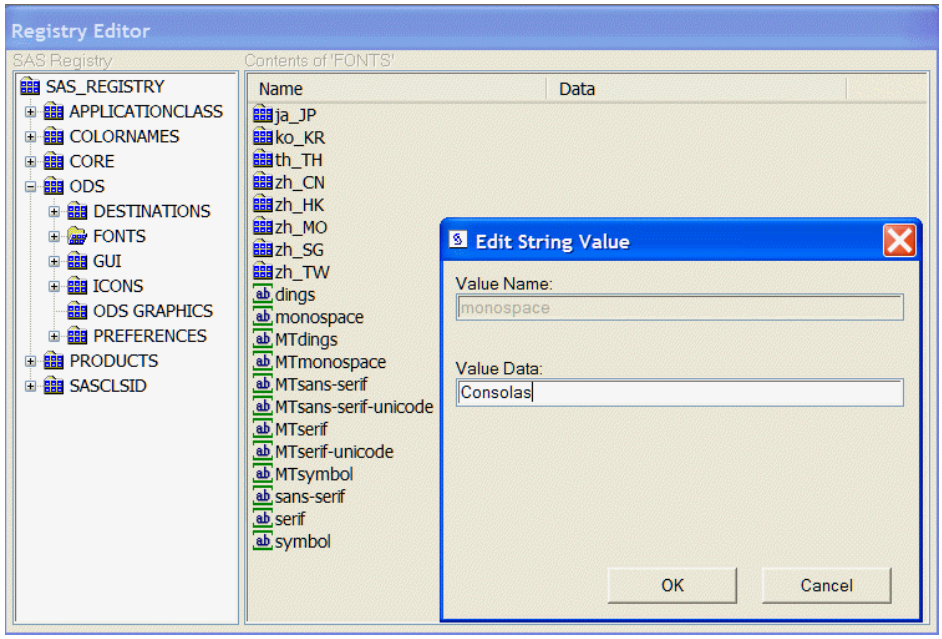
注意:

レジストリ値を変更する前に、Sasuser から regstry.sas7bitm ファイルを必ずバックアップします。

1. **レジストリエディタ**ウィンドウの左ペインで、変更するキーをクリックします。キーに含まれる値が右ペインに表示されます。
2. 値をダブルクリックします。

   変更する値に種類に従って、複数の種類のウィンドウがレジストリエディタに表示されます。

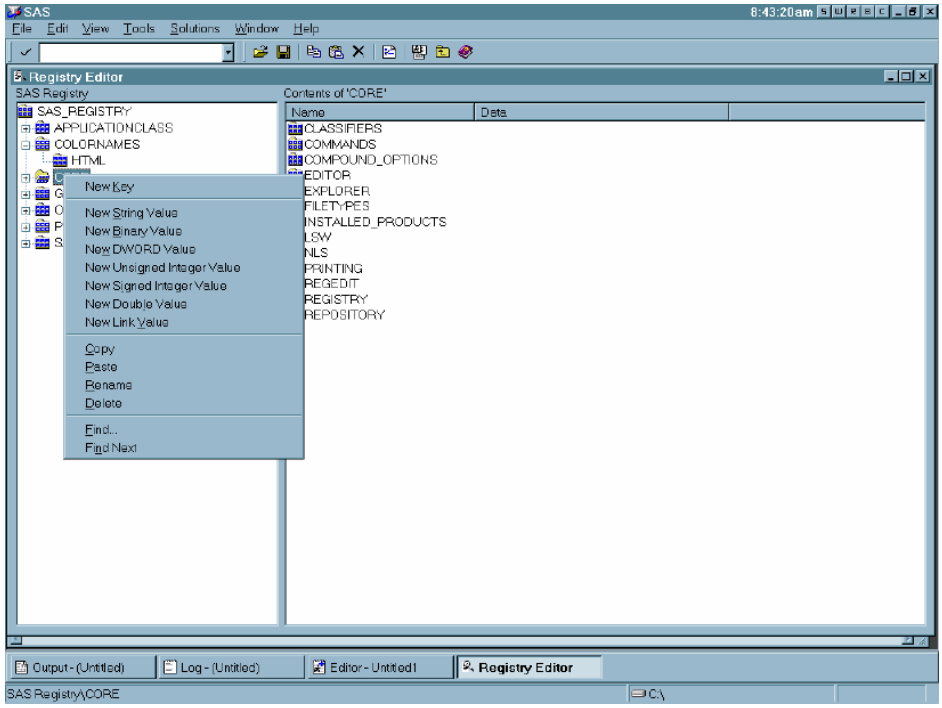
図 14.3 SAS レジストリ内の値を変更するウィンドウの例



SAS レジストリへの新しい値またはキーの追加

1. SAS レジストリエディタで、値追加先のキーを右クリックします。
2. ポップアップメニューで、作成する種類に対応する**作成**メニュー項目を選択します。
3. 表示されたウィンドウに、新しいキーまたは値を入力します。

図 14.4 レジストリエディタ(新しいキーおよび値を追加するためのポップアップメニューが表示された状態)



SAS レジストリからの項目の削除

SAS レジストリエディタ内で次の操作を実行します。

1. 削除する項目を右クリックします。
2. ポップアップメニューから**削除**を選択します。
3. 削除を確定します。

SAS レジストリ内の項目名の変更

SAS レジストリエディタ内で次の操作を実行します。

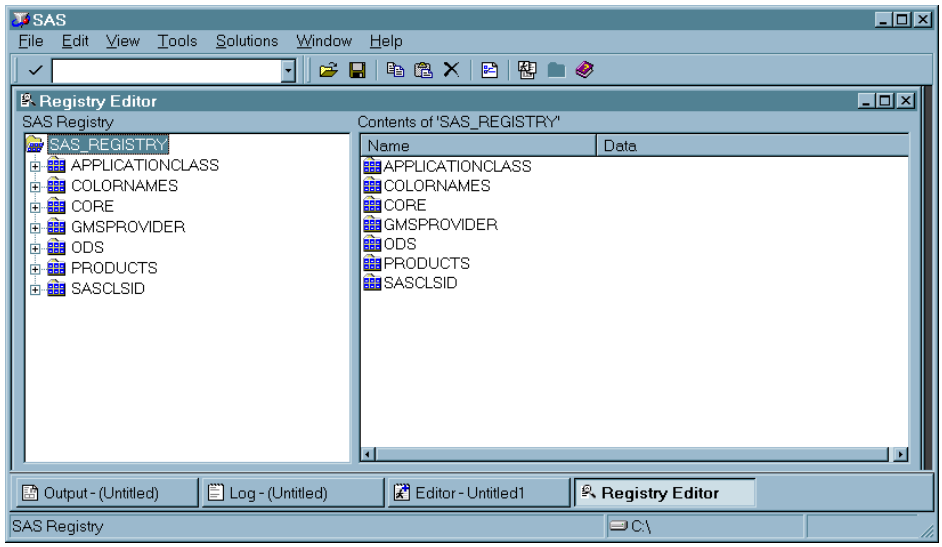
1. 名前変更する項目を右クリックします。
2. コンテキストメニューで**名前の変更**を選択し、新しい名前を入力します。
3. **OK** をクリックします。

Sasuser レジストリ項目と Sashelp レジストリ項目の個別表示

レジストリエディタを開いた後で、デフォルトの表示を変更できます。デフォルト表示では、格納場所に関係なくレジストリの内容が表示されます。他方のレジストリビューには、レジストリエディタの左ペインで Sasuser 項目と Sashelp 項目が別々のツリーで表示されます。

1. **ツール** ⇨ **オプション** ⇨ **レジストリエディタ**を選択すると、**レジストリビューの選択**グループボックスが表示されます。
2. **すべて表示する**を選択して、レジストリエディタの左ペインに Sasuser 項目と Sashelp 項目を別々に表示します。
   - レジストリの Sashelp 部分は、左ペインの HKEY_SYSTEM_ROOT フォルダの下にリストされます。
   - レジストリの Sasuser 部分は、左ペインの HKEY_USER_ROOT フォルダの下にリストされます。

図 14.5 レジストリエディタ(重ね合わせで表示モード)



レジストリファイルのインポート

バックアップレジストリファイルを復元する際、通常、レジストリファイルや SASXREG ファイルをインポートします。レジストリファイルには、完全なレジストリを格納することも、レジストリの一部を格納することもできます。

SAS レジストリエディタを使用してレジストリをインポートするには、次の操作を実行します。

1. **ファイル** ⇨ **レジストリファイルのインポート**を選択します。
2. **開く**ウィンドウで、インポートする SASXREG ファイルを選択します。

注: バックアップレジストリファイルを最初に作成するには、REGISTRY プロシジャを使用するか、レジストリエディタの**レジストリファイルのエクスポート**メニューを選択します。

レジストリファイルのエクスポート

バックアップレジストリファイルを用意する際、通常、レジストリファイルや SASXREG ファイルをエクスポートします。完全なレジストリでもレジストリの一部でもエクスポートできます。

SAS レジストリエディタを使用してレジストリをエクスポートするには、次の操作を実行します。

1. レジストリエディタの左ペインで、SASXREG ファイルにエクスポートするキーを選択します。

   レジストリ全体をエクスポートするには、最上位のキーを選択します。
2. **ファイル** ⇨ **レジストリファイルのエクスポート**を選択します。
3. **名前を付けて保存**ウィンドウで、エクスポートファイルに名前を付けます。
4. **保存**をクリックします。

レジストリの設定

ユニバーサル印刷の設定

ユニバーサルプリンタを設定するには、PRTDEF プロシジャまたは**印刷設定**ウィンドウを使用します。REGISTRY プロシジャを使用すると、プリンタ定義をバックアップしたり、SASXREG ファイルからプリンタ定義を復元したりできます。それ以外の方法でレジストリ値を直接変更する作業は、SAS テクニカルサポートの指示に従った場合にのみ可能です。

SAS エクスプローラの設定

エクスプローラを設定する場合は**エクスプローラオプション**ウィンドウを使用するのが最も良い方法ですが、レジストリエディタを使用して SAS レジストリの現在のエクスプローラ設定を表示することもできます。エクスプローラオプションウィンドウは、エクスプローラ内の**ツール** ⇨ **オプション** ⇨ **エクスプローラ**メニューから使用可能です。エクスプローラ構成データはすべて CORE\Explore の下のレジストリに格納されています。最も一般的なエクスプローラ構成データの場所を次の表に示します。

表 14.1 一般的なエクスプローラ構成データのレジストリの場所

レジストリキー	構成するエクスプローラの部分
CORE\EXPLORER\CONFIGURATION	起動時に初期化されるエクスプローラの部分。
CORE\EXPLORER\MENUS	エクスプローラに表示されるコンテキストメニュー。
CORE\EXPLORER\KEYEVENTS	3270 インターフェイスの有効なキーイベント。このキーは、メインフレームプラットフォームでのみ使用されます。
CORE\EXPLORER\ICONS	エクスプローラで表示されるアイコン。アイコン値が–1 の場合、アイコンは表示されません。
CORE\EXPLORER\NEW	このサブキーは、エクスプローラの**ファイル** ⇨ **新規**メニューで指定可能なオブジェクトを制御します。

SAS レジストリを使用したライブラリ参照名とファイルショートカットの設定

ライブラリの作成ウィンドウまたは**ファイルショートカットの作成**ウィンドウを使用してライブラリ参照名(libref)またはファイル参照名(fileref)を作成した場合、この 2 つのウィンドウのいずれかの**起動時に有効**チェックボックスをオンにすると、以降の作業で使用できるようにこれらの参照が格納されます。

ライブラリ参照名(libref)とファイル参照名(fileref)は、**起動時に有効**チェックボックスをクリックすると保存され、SAS レジストリに格納されます。次のように、変更または削除できます。

“起動時に有効”なライブラリ参照名の削除
CORE\OPTIONS\LIBNAMES\“your libref”の下の対応キーを削除すると、レジストリエディタを使用して“起動時に有効”なライブラリ参照名を削除できます。ただし、ライブラリ参照名を削除する最も適切な方法は、SAS エクスプローラを使用することです。SAS エクスプローラでライブラリ参照名を削除すると、このキーはレジストリから自動的に削除されます。

“起動時に有効”なファイルショートカットの削除
レジストリエディタを使用すると、CORE\OPTIONS\FILEREFS*your fileref* の下にある対応するキーを削除することで、“起動時に有効”なファイルショートカットを削除できます。ただし、ライブラリ参照名を削除する最も適切な方法は、SAS エクスプローラを使用することです。SAS エクスプローラでファイルショートカットを削除すると、このキーはレジストリから自動的に削除されます。

“起動時に有効”なファイルショートカットをサイトデフォルトとして作成
サイト管理者は、サイトの全ユーザーが使用可能なファイルショートカットを作成できます。これを実行するには、最初に、Sasuser レジストリでファイルショートカットのバージョンを作成します。その後、それが Sashelp レジストリで使用できるように変更します。

注: SAS レジストリの Sashelp 部分に書き込むための特別なアクセス許可が必要です。

1. **DMFILEASSIGN** コマンドを入力します。

   これにより、**ファイルショートカットの作成**ウィンドウが表示されます。

2. 使用するファイルショートカットを作成します。
3. **起動時に有効**チェックボックスをクリックします。
4. **OK** をクリックします。
5. ファイルショートカットが正常に作成されたことを確認して、**REGEDIT** コマンドを入力します。
6. CORE\OPTIONS\FILEREFS*your fileref* というキーを検索して選択します。
7. **ファイル** ⇨ **レジストリファイルのエクスポート**を選択し、ファイルをエクスポートします。
8. エクスポートしたファイルを編集し、HKEY_USER_ROOT のすべてのインスタンスを HKEY_SYSTEM_ROOT に置き換えます。
9. サイトの Sashelp に変更を適用するには、PROC REGISTRY を使用します。

   次のコードはファイルをインポートします。

```
proc registry import="yourfile.sasxreg" usesashelp;
run;
```

“起動時に有効”なライブラリをサイトデフォルトとして作成

サイト管理者は、サイトの全ユーザーが使用可能なライブラリを作成できます。これを実行するには、ライブラリ定義の Sasuser バージョンを Sashelp に移行する必要があります。

注: SAS レジストリの Sashelp 部分に書き込むための特別なアクセス許可が必要です。

1. **dmlibassign** コマンドを入力します。

   これにより、**ライブラリの作成**ウィンドウが表示されます。

2. 使用するライブラリ参照名を作成します。
3. **起動時に有効**チェックボックスをクリックします。
4. **起動時に有効**チェックボックスをクリックします。
5. **OK** をクリックします。
6. ライブラリが正常に作成されたことを確認したら、**REGEDIT** コマンドを発行します。
7. CORE\OPTIONS\LIBNAMES*your libref* というレジストリキーを検索して選択します。
8. **ファイル** ⇨ **レジストリファイルのエクスポート**を選択します。

   名前を付けて保存ウィンドウが表示されます。

9. レジストリファイルを格納する場所を選択します。
10. レジストリファイルのファイル名を**ファイル名**フィールドに入力します。
11. **保存**をクリックしてファイルをエクスポートします。
12. ファイルを右クリックし、**NOTEPAD で編集**を選択してファイルを編集します。
13. エクスポートしたファイルを編集し、“HKEY_USER_ROOT”のすべてのインスタンスを“HKEY_SYSTEM_ROOT”に置き換えます。

14. サイトの Sashelp に変更を適用するには、PROC REGISTRY を使用します。次のコードはファイルをインポートします。

```
proc registry import="yourfile.sasxreg" usesashelp;
run;
```

SAS レジストリを使用したライブラリ参照名(Libref)の問題の解決

ライブラリ参照名(libref)は SAS レジストリに格納されます。.以前は機能していたライブラリ参照名に不具合が発生することもあります。場合によっては、レジストリを編集することが問題解決の最も近道になります。このセクションでは、ライブラリ参照名の欠損や障害の修復に関して説明します。

SAS レジストリに格納されている永久ライブラリ参照名が起動時に失敗した場合、次の注記が SAS ログに表示されます。

```
NOTE: One or more library startup assignments were not restored.
```

次のエラーは、ライブラリの割り当てに関する問題で一般的な原因です。

- SAS レジストリでライブラリ参照名の割り当てに必要なフィールド値が失われています。
- SAS レジストリでライブラリ参照名の割り当てに必要なフィールド値が無効です。たとえば、ライブラリ名は 8 文字に制限されており、エンジン値は実際のエンジン名に一致している必要があります。
- ライブラリ参照名の暗号パスワードが SAS レジストリで変更されています。

注: **ライブラリの作成**ウィンドウでライブラリ参照名を追加することもできます。このウィンドウを開くには、ツールバーに DMLIBASSIGN と入力するか、**ファイル** ⇨ **新規**を**エクスプローラ**ウィンドウで選択します。

注意:

SAS レジストリエディタで、多くのライブラリ参照名割り当てエラーを修正できます。 SAS レジストリエディタのライブラリ参照名に習熟していない場合は、テクニカルサポートにお問い合わせください。SAS レジストリエディタではエラーが簡単に起こる可能性があり、起動時にライブラリを割り当てられなくなるおそれがあります。

SAS レジストリエディタを使用してライブラリ参照名エラーを修正するには、次の操作を実行します。

1. **ソリューション** ⇨ **アクセサリ** ⇨ **レジストリエディタ**を選択するか、**REGEDIT** コマンドを発行して、レジストリエディタを開きます。
2. 動作環境に応じて次のいずれかのパスを選択し、必要に応じてキーとキー値を変更します。

   ```
   CORE\OPTIONS\LIBNAMES
   ```

   または

   ```
   CORE\OPTIONS\LIBNAMES\CONCATENATED
   ```

注: これらの修正は、永久ライブラリ参照名の場合にのみ可能です。つまり、**ライブラリの作成**ウィンドウまたは**ファイルショートカットの作成**ウィンドウで起動時に作成されたライブラリ参照名の場合にのみ可能です。

たとえば、永久連結ライブラリのキーが正の整数以外に名前変更されていると判断した場合は、準拠するようにキーを再び名前変更できます。処理を開始するには、キーを選択し、ポップアップメニューで **Rename** を選択します。

15 章

SAS を用いた印刷

ユニバーサル印刷

ユニバーサル印刷について

ユニバーサル印刷とは、さまざまなドキュメントとグラフィック出力の形式を作成して対話型とバッチ両方の印刷機能を提供するシステムです。たとえば、HTML、PDF、ドキュメントおよび PNG、GIF、SVG グラフィックの作成にユニバーサル印刷が使用できます。サポートされているドキュメントやグラフィックタイプの完全なリストについては、表 15.1 (228 ページ)を参照してください。

ユニバーサル印刷では、プリンタや印刷プレビューアを定義したり、印刷される出力を制御するオプションを設定したりできます。さまざまなドキュメントとグラフィック出力タイプの作成に加え、出力をプリンタに送ることができます。

Windows 固有
デフォルトでは、Windows 動作環境は、ユニバーサル印刷ではなく Windows 印刷を使用します。Windows でユニバーサル印刷を使用する方法については、"ユニバーサル印刷のインターフェイスとデフォルトの印刷環境の設定" (227 ページ)を参照してください。

SAS はユニバーサル印刷サービスを通じてすべての印刷を送信します。ユニバーサル印刷機能はシステムオプションで制御するので、バッチモードでもさまざまな印刷機能を制御できます。SAS システムオプションの詳細については、"ユニバーサル印刷を制御するシステムオプション" (259 ページ)を参照してください。

注: ユニバーサル印刷を導入する前、SAS では、フォームという印刷ジョブ用ユーティリティをサポートしていました。フォーム印刷は、ウィンドウ環境のメニューから**ファイル** ⇨ **印刷設定**を選択すると、現在でも使用できます。その後、**フォームを使用する**チェックボックスを選択します。この操作により、ユニバーサル印刷メニューと機能は無効になります。詳細については、"フォーム印刷" (267 ページ)を参照してください。

ユニバーサル印刷のインターフェイスとデフォルトの印刷環境の設定

UNIX および z/OS でのユニバーサル印刷

UNIX および z/OS 動作環境では、SAS の起動時にユニバーサル印刷が有効になります。それ以上のアクションは必要ありません。

Windows でのユニバーサル印刷

Windows では、SAS の起動時に Windows 印刷が有効になります。ユニバーサル印刷を Windows で使用するには、ユニバーサル印刷環境、メニューおよびダイアログボックスを有効にするように UNIVERSALPRINT システムオプションを設定し、印刷デフォルトを設定する必要があります。SAS ウィンドウ環境を使用する場合は、UPRINTMENUSWITCH システムオプションを使用して**ファイル**メニューで印刷コマンドを有効化することもできます。これらのオプションは、SAS 構成ファイルか、SAS 起動時にのみ設定可能です。SAS 起動後にユニバーサル印刷メニューおよびダイアログボックスを有効または無効にすることはできません。

SAS 起動時に次のシステムオプションを含めます。

```
-uprint -uprintmenuswitch
```

UPRINT は、UNIVERSALPRINT システムオプションの別名です。

UPRINT オプションのみで SAS を開始する場合、デフォルトでは開いている、HTML の出力先を閉じる必要があります。PRINTERPATH=オプションを使用して、出力形式タイプを指定します。次に、ODS PRINTER ステートメントと ODS PRINTER CLOSE ステートメントを使用して、実行したいコードの前後を囲みます。次に例を示します。

```
ods html close;
options printerpath=pdf;
ods printer;
   proc print data=sashelp.class;
   run;
ods printer close;
```

デフォルトプリンタへ戻す

PRINTERPATH=システムオプションを使用してプリンタを指定すると、印刷ジョブはユニバーサル印刷で制御されます。デフォルトのユニバーサルプリンタ(PostScript プリンタ)に戻すには、PRINTERPATH=オプションを Null 値(間にスペースを入れずに二重引用符を 2 つ)に設定します。

```
options printerpath="";
```

Windows では、ユニバーサル印刷が有効でない場合、PRINTERPATH=を Null 値に設定することで Windows 印刷に印刷が戻ります。

ユニバーサル印刷の出力形式

印刷ジョブをプリンタに送信することに加え、異なる種類のプリンタやソフトウェアプログラムで幅広く認識されている外部ファイルに直接出力することもできます。ユニバーサル印刷を使用すると、次の一般的なファイルタイプを生成できます。

表 15.1 *使用可能な印刷出力形式*

タイプ	フルネーム	説明
GIF	Graphics Interchange Format	グラフィックデータをオンラインで送信したりやり取りしたりするための画像形式です。サイズが比較的小さく移植性が高いので、World Wide Web で画像を表示するために幅広く使用されています。
EMF	Enhanced Metafile Format	カラー、スケーラブル、デバイス非依存のグラフィックを作成するためにグラフィック描画コマンド、構成プロパティ、グラフィックオブジェクトをまとめたメタファイル形式です。EMF をサポートするアプリケーションは Windows で実行されています。ユニバーサル印刷は、メタファイル形式の EMF、EMFPlus および EMFDual レベルを現在サポートしています。EMF ユニバーサルプリンタは、メタファイル形式 EMFPlus レベルを使用します。これがデフォルトの EMF プリンタです。
PCL	Printer Control Language	Hewlett-Packard によって開発され、さまざまな印刷デバイスの幅広いプリンタ機能を制御するためにアプリケーションが使用する言語です。ユニバーサル印刷は現在、PCL4、PCL5、PCL5e、PCL5c レベルの言語をサポートしています。

タイプ	フルネーム	説明
PDF	Portable Document Format	整形済みドキュメントを表示、印刷するために、Adobe Systems によって開発されたファイル形式です。PDF 形式でファイルを表示するには、Adobe Systems が配布しているフリーアプリケーション Adobe Reader が必要です。 *注:* Adobe Acrobat は、ユニバーサル印刷で PDF ファイルを作成する場合は不要です。
PNG	Portable Network Graphics	古い単純な GIF 形式とより複雑な TIFF 形式のかわりとして設計された画像形式。GIF の場合と同様、PNG は主に Web で画像を表示するために使用されます。Web では、GIF よりも PNG に大きな利点があります。それは、ガンマ補正、2 次元インターレース、可変透過度(アルファチャネル)、解像度の設定、256 色以上に対応していることです。
PS	PostScript	Adobe Systems で開発されたページ記述言語です。これがデフォルトのユニバーサルプリンタです。
SVG	Scalable Vector Graphics	2 次元グラフィックとグラフィカルアプリケーションを XML で記述するための言語であるベクトル形式です。
TIFF	Tagged Image File Format	単一ファイルの中でイメージとデータの両方をサポートする Adobe ラスタイメージ形式。TIFF ユニバーサルプリンタは、RGBA カラー印刷および透過性をサポートします。TIFFk ユニバーサルプリンタは、CYMK カラー印刷をサポートします。

前述の形式のいずれかで出力を作成するには、PRINTERPATH=システムオプションの値をユニバーサルプリンタに設定するか、ODS ステートメントを使用します。PRINTERPATH=システムオプションが、ファイルに出力をするプリンタに設定されている場合、デフォルトのファイル名は sasprt.*extension* です。*extension* は、プリンタ形式タイプです。たとえば、sasprt.pdf、sasprt.emf、sasprt.png、sasprt.gif などです。ファイルは現在のディレクトリに上書きされます。

ファイルの場所や名前を変更するには、PRINTERPATH=システムオプションを使用できます。次に例を示します。

```
options printerpath=(svg out);
filename out 'c:\myimages\graph1.svg';
```

ユニバーサルプリンタとプリンタプロトタイプの表示

SAS は、独自のプリンタを作成するために使用できるユニバーサルプリンタとプリンタのプロトタイプを提供します。**印刷**ダイアログボックスから使用可能なプリンタのリストにアクセスできます。QDEVICE プロシジャを使用してプリンタのデータセットを作成し、PRINT プロシジャを使用してプリンタ情報を印刷することもできます。

プリンタの表を作成し、各プリンタの説明付きでリストを印刷するには、次のコードをサブミットします。

```
proc qdevice out=printers;
   printer _all_;
run;

proc print data=printers;
   var name desc;
```

```
run;
```

詳細については、“QDEVICE Procedure” (*Base SAS Procedures Guide*)を参照してください。

プリンタプロトタイプのリストを SAS ログに出力するには、次の SAS プログラムをサブミットします。

```
filename registry temp;
proc printto log=registry;
run;

proc registry list keysonly levels=1 startat="core\printing\prototypes";
proc printto;
run;

data protypes;
   keep prototype;
   infile registry lrecl=300 pad;
   length line $300;
   input line $300.;
   if substr(line,1,1) = "["
     then do;
        prototype = strip(substr(line,2,length(line)-2);
        if index(prototype,'core\printing\prototypes') ne 0
          then delete;
        else
          output;
      end;

run;

proc print label;
   label prototype = "Prototype";
run;
```

詳細については、“REGISTRY Procedure” (*Base SAS Procedures Guide*)を参照してください。

ユニバーサルプリンタ設定の表示

QDEVICE プロシジャまたは**印刷**ダイアログボックスを使用すると、ユニバーサルプリンタの設定を参照できます。QDEVICE プロシジャを使用してプリンタ設定を表示するには、次のコードをサブミットします。

```
proc qdevice;
printer printer-name;
run;
```

GIF プリンタのプリンタ設定は次のようになります。

```
   19   proc qdevice; 20   printer gif; 21    run; Name:GIF Description:Graphics Interchang
```

QDEVICE プロシジャでは、すべてのプリンタ設定をレポートしません。レポート可能なプリンタ設定の説明については、“QDEVICE Procedure” (*Base SAS Procedures Guide*)を参照してください。

ユニバーサル印刷プリンタ設定の変更

ユニバーサルプリンタのダイアログボックスでプリンタ設定を変更するには、SAS システムオプションを設定するか、PRTDEF プロシジャを使用します。次のトピックを参照してください。

- “ウィンドウ環境を用いたユニバーサル印刷の設定” (242 ページ)
- “ユニバーサル印刷を制御するシステムオプション” (259 ページ)
- “PRTDEF プロシジャを用いたユニバーサルプリンタの管理” (261 ページ)

ユニバーサル印刷と ODS

ODS PRINTER ステートメントは、UNIVERSALPRINT または NOUNIVERSALPRINT システムオプションが設定されている場合にユニバーサル印刷を使用できます。ODS PRINTER ステートメントで使用される PRINTER の出力先は、“ODS PRINTER Statement ” (*SAS Output Delivery System: User's Guide*)に記載されています。

ODS (Output Delivery System)は、次の ODS ステートメントでユニバーサル印刷を使用します。

表 15.2 *ユニバーサル印刷を使用する ODS ステートメント*

ODS ステートメント	説明
ドキュメントの出力形式	
ODS PRINTER PRINTER=オプション	選択したプリンタを使用します。
ODS PDF ステートメント	ユニバーサル印刷 PDF プリンタを使用します。*
ODS PS ステートメント	ユニバーサル印刷 PostScript Level 1 プリンタを使用します。
ODS PCL ステートメント	ユニバーサル印刷 PCL5 プリンタを使用します。
グラフィックの出力形式	
ODS LISTING	選択したプリンタを使用します。
ODS HTML	ODS Graphics および SAS/GRAPH と一緒に使用します。
ODS HTML5	ODS Graphics および SAS/GRAPH と一緒に使用します。
ODS RTF	ODS Graphics および SAS/GRAPH と一緒に使用します。

ODS ステートメント	説明
ODS EPUB	ODS Graphics および SAS/GRAPH と一緒に使用します。

* PDF ユニバーサルプリンタで作成されたグラフのドリルダウン領域を作成するには、SAS/GRAPH をインストールする必要があります。詳細については、“Adding Drill-Down Graphs in Your PDF File” (*SAS/GRAPH: Reference*)を参照してください。

Windows 固有
Windows 動作環境では、ODS PRINTER 出力先は Windows システムプリンタを使用します。ただし、SAS が UNIVERSALPRINT システムオプションで起動している場合や、PRINTERPATH=システムオプションでプリンタを指定した場合は除きます。ユニバーサル印刷が Windows で有効になっている場合、SAS は、Windows システムプリンタの使用をオーバーライドするので、ODS ではユニバーサル印刷が使用されます。Windows 印刷に戻すには、PRINTERPATH=システムオプションを PRINTERPATH="" (間にスペースを入れずに二重引用符を 2 つ)として Null 文字列に設定します。

ODS の詳細については、*SAS Output Delivery System: User's Guide* を参照してください。

ユニバーサル印刷ドキュメントのページの向きの指定

ユニバーサルプリンタで作成された複数ページドキュメントのページごとにページの向きを指定できます。ODS LISTING、PCL、PDF、PRINTER、および PS 出力先用に作成されたドキュメントのページの向きも指定できます。

ORIENTATION=システムオプションには、PORTRAIT、LANDSCAPE、REVERSEPORTRAIT、REVERSELANDSCAPE の 4 つの値があります。ドキュメントページの向きを変えるには、ページの向きを変更するための出力を作成するステップ間で、ORIENTATION=システムオプションを使用して OPTIONS ステートメントを指定します。

注: EMF、GIF、PNG、および TIFF ユニバーサルプリンタは複数ページドキュメントをサポートしていません。また、これらのプリンタは REVERSELANDSCAPE および REVERSEPORTRAIT の向きもサポートしていません。

次の例では、4 ページの SVG ドキュメントを作成します。ドキュメントのページごとに、向きを縦または横に変更しています。OPTIONS ステートメントは強調表示されています。

```
options nodate nonumber;
ods printer printer=svgview file='orientation.svg' style=Ocean;
title 'Demonstration of Page Orientation Changes in a Document';
footnote 'PROC SGPLOT in Landscape Orientation';
options orientation=landscape;
proc sgplot data=sashelp.class;
vbar age;
run;

options orientation=portrait;
footnote 'PROC PRINT in Portrait Orientation';
proc print data=sashelp.class;
run;

options orientation=landscape;
footnote 'PROC SGSCATTER in Landscape Orientation';
```

```
proc sgscatter data=sashelp.cars;
matrix mpg_city enginesize horsepower /
   diagonal=(histogram kernel);
run;

options orientation=portrait;
footnote 'PROC MEANS in Portrait Orientation';
proc means data=sashelp.cars n mean;
   var  mpg_city enginesize horsepower;
run;

ods printer close;
```

次の出力は、ドキュメントの 3 ページ目と 4 ページ目を示します。3 ページ目は横向き、4 ページ名は縦向きです。

図 15.1 *SVG ドキュメントの 3 ページ目(横向き)*

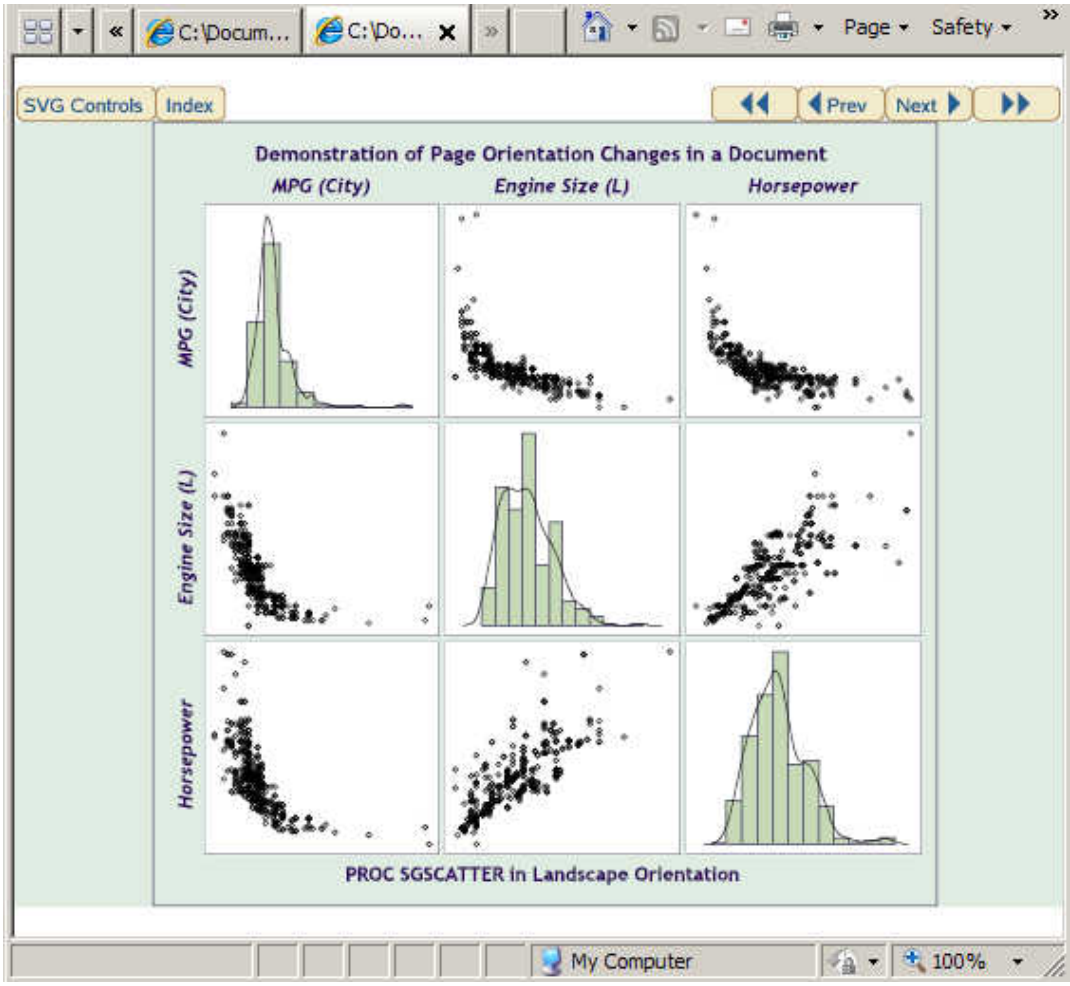
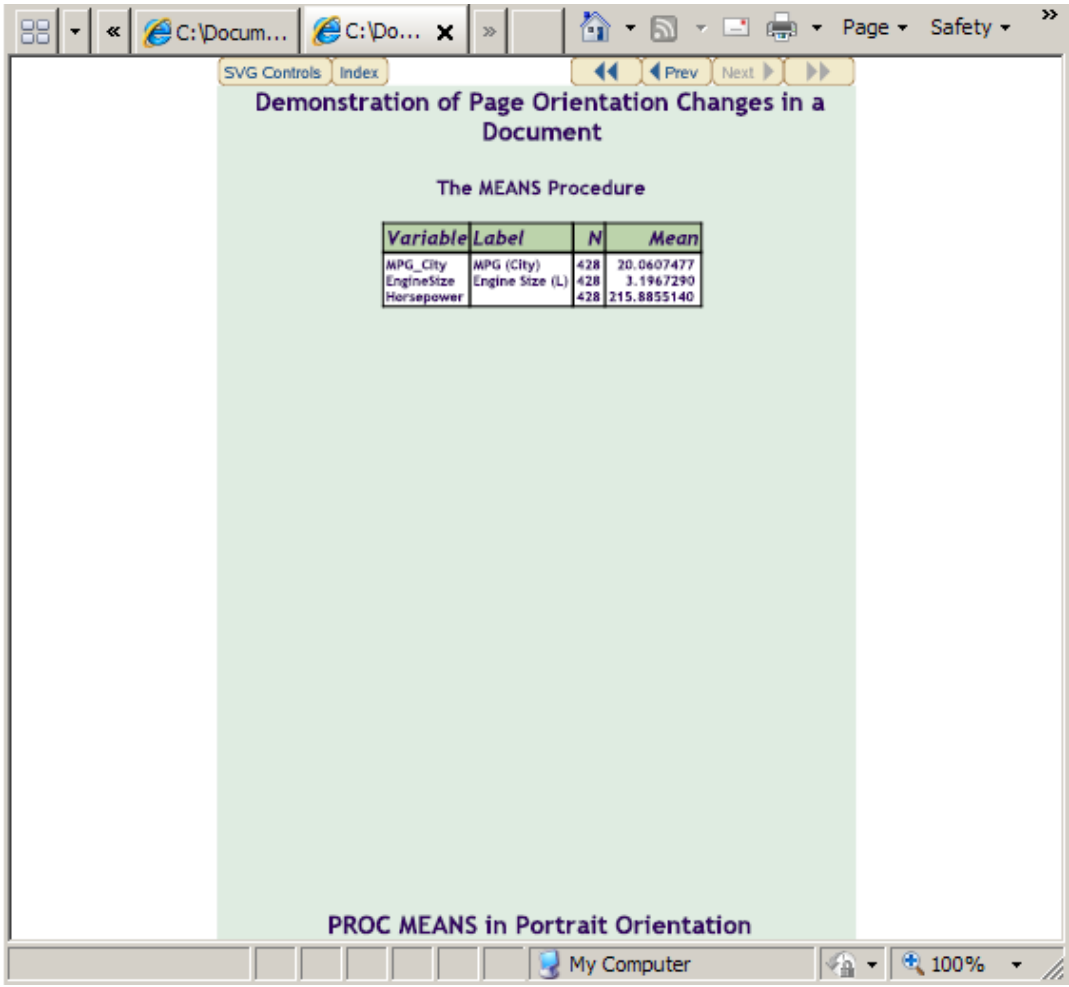


図 15.2 SVG ドキュメントの 4 ページ目(縦向き)



詳細については、“ORIENTATION= System Option” (*SAS System Options: Reference*)を参照してください。

ユニバーサルプリンタのカラーサポート

ユニバーサルプリンタとサポートされている色空間

すべてのユニバーサルプリンタは 24 ビット RGB カラーをサポートしています。ほとんどのプリンタは現在、32 ビット CMYK カラーまたは 32 ビット RGBA (透過)カラーをサポートしています。次の表に、ユニバーサルプリンタとそれぞれがサポートする色をリストします。

表 15.3 ユニバーサルプリンタのカラーサポート

ユニバーサルプリンタ	カラーサポート	透過をサポート
EMF	RGBA	あり

ユニバーサルプリンタ	カラーサポート	透過をサポート
EMFDual	EMF ビューアが EMFPlus 形式をサポートする場合は、RGBA カラーがサポートされます。 EMF ビューアが EMFPlus 形式をサポートしない場合は、ビットマップ画像の RGBA カラーとベクター要素の RGB カラーがサポートされます。	あり あり(ビットマップ画像のみ)
SASEMF	ビットマップ画像の場合のみ RGBA。ベクター要素の場合は RGB。	あり(ビットマップ画像のみ)
GIF	RGBA	あり
PCL5c*	RGBA	あり
PDF	CMYK および RGBA	あり(RGBA カラーのみ)
PNG	RGBA	あり
PostScript	CMYK および RGB GIF 画像の場合は RGBA **	なし なし
SVG	RGBA	あり
TIFF	RGBA	あり
TIFFk	CYMK	なし

* PCL4 および PCL5 ユニバーサルプリンタはモノクロ印刷のみをサポートしています。

** PostScript ユニバーサルプリンタは、RGBA カラーを認識しますが、透過性はサポートしていません。

CYMK、RGB、RGBA カラーの詳細については、“CMYK カラー” (235 ページ)および “RGB および RGBA カラー” (237 ページ)を参照してください。

CMYK カラー

CMYK カラー設定は、シアン、マゼンタ、黄、ブラックのインク量を指定するために 8 つの 16 進文字(0 - 255)を指定します。プリンタのパントーン色見本帳で、プリンタの CMYK 値を見つけます。EMF プリンタで CMYK カラーなど、サポートされていない色を指定した場合、色はサポートされている色に変換されます。

色を設定できる場所であればどこでも、CMYK カラーを指定できます(たとえば、PROC PRINT ステートメントの STYLE オプションまたは TITLE ステートメントで)。

16 進数を CMYK または K で始めます。SAS で設定可能な CMYK カラーの例を次に示します。

表 15.4 *CMYK カラーの例*

16 進数表現	色
cmykFF000000	シアン

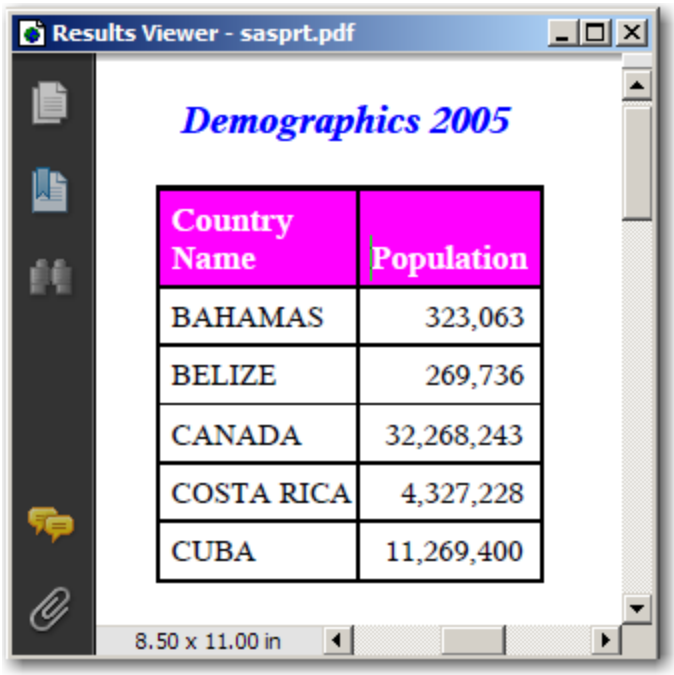
16 進数表現	色
k00FF0000	マゼンタ
cmyk0000FF00	黄
kFFFF0000	青
cmykFF00FF00	緑
k00FFFF00	赤
cmykFFFFFF00	シアン、マゼンタ、黄を使用したプロセスブラック
k000000FF	黒

16 進数の最初のバイトはシアンを表します。2 番目のバイトはマゼンタを表します。3 番目のバイトは黄色を表します。4 番目のバイトは黒を表します。

次の例では、STYLE オプションを使用し、列の背景色をマゼンタに設定し、前景色を白に設定します。次の TITLE ステートメントは、出力タイトルを青に設定します。

```
options obs=5 nodate;
ods html close;
ods pdf;
proc print data=sashelp.demographics label
           style(header)={background=cmyk00ff0000 foreground=k00000000} noobs;
   var name pop;
   label name=Country Name pop=Population;
   title color=kffff0000 'Demographics 2005';
run;
ods pdf close;
ods html;
```

図 15.3 STYLE オプションで指定した CMYK カラー



注: TIFFk ユニバーサルプリンタを使用する場合、UPRINTCOMPRESSION システムオプションを指定して、TIFF ファイルが大きくなりすぎないようにします。

RGB および RGBA カラー

RGB カラーと RGBA カラーは、赤、緑、青を異なる比率で組み合わせて色を作ります。A はアルファチャネルで、不透明度のパーセントを表します。

RGB カラーは、3 組の 16 進数(00 - FF)で指定します。各 16 進数は、赤、緑、青がどの程度含まれるかを示します。RGBA カラーは、不透明度を示すアルファチャネルにさらに 16 進数を加えたものです。FF は不透明で、00 は透明です。RGB および RGBA カラーの指定では、最初の 16 進数は赤、2 番目は緑、3 番目は青です。RGBA カラーでは、4 番目の 16 進数はアルファチャネルの指定です。

RGB カラーと RGBA カラーは、色を設定できる場所であればどこでも(たとえば、SGPLOT プロシジャの VBAR ステートメントまたは TITLE ステートメントのオプションで)指定できます。RGB カラーの場合、16 進数を CX で始めます。RGBA カラーの場合、16 進数を RGBA または A で始めます。

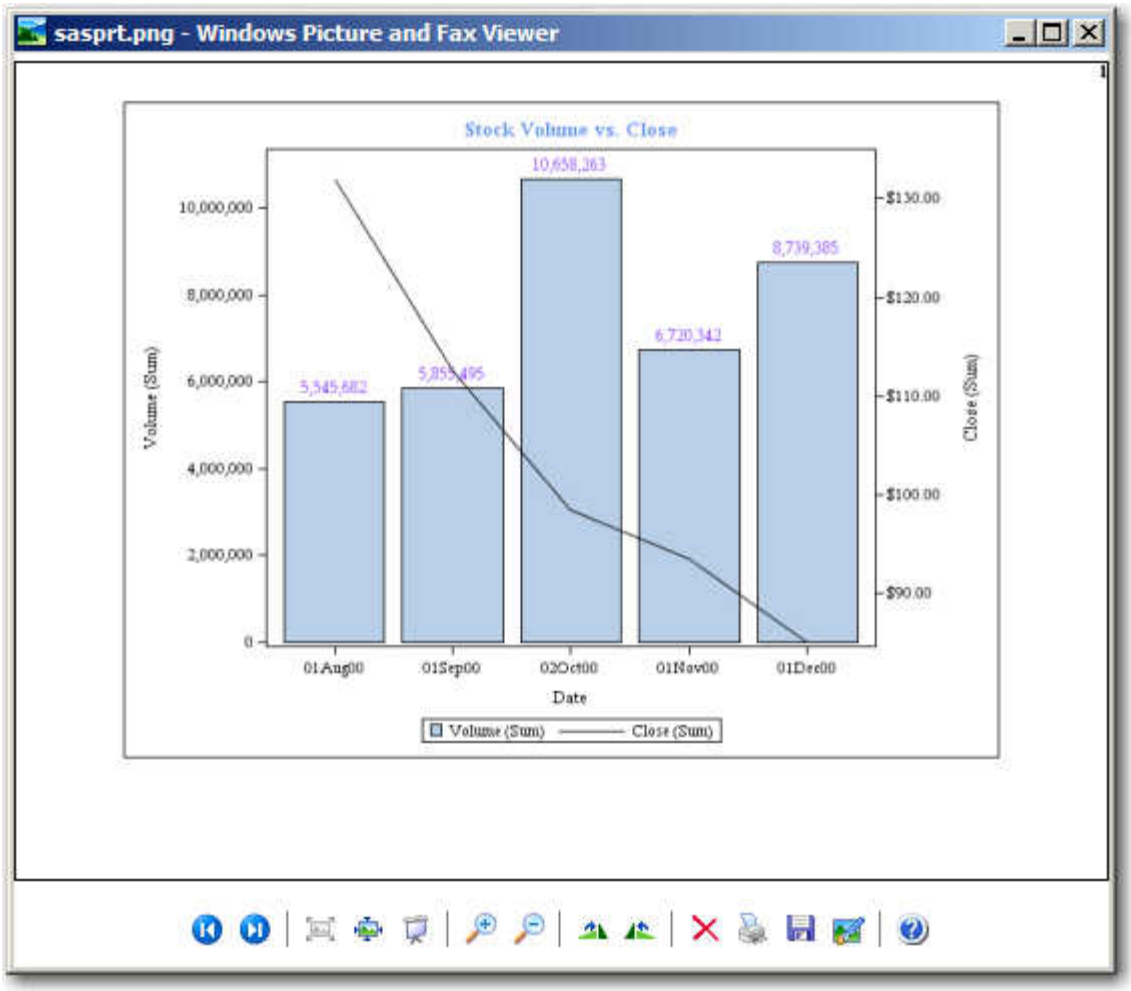
次の SGPLOT プロシジャは、RGBA カラーを使用して棒のラベルを作成します。

```
ods html close;
ods printer printer=png;
proc sgplot data=sashelp.stocks (where=(date >= "01jan2000"d
                                 and date <= "01jan2001"d
                                 and stock = "IBM"));
   title color=a6495edff "Stock Volume vs. Close";
   vbar date / response=volume
               datalabel
               datalabelattrs=(color=a8a44ff8a size=10);
   vline date / response=close y2axis;
run;
title;
ods printer close;
```

```
ods html;
```

次に、棒のラベル付きの PNG ファイルを示します。

図 15.4 棒のラベルに指定された RGBA カラー



例:RGBA カラーを使用した静的で異なる表の背景色

この例のプログラムでは、次のことを実行します。

- DATA _NULL_ステートメントを使用して PCT.形式を作成します。DATA ステップは$3,000.00 の給与範囲を定義し、給与範囲ごとの RGBA カラー値を計算します。CALL EXECUTE ステートメントは、FORMAT プロシジャコードを生成時に出力するために使用します。
- データセットを作成します。
- PRINT プロシジャは、表見出しの背景で RGBA カラー値を使用し、PCT.形式を使用して給与変数をフォーマットします。

```
options nodate;

/* Create the PCT format.                                  */
/* The color variable is a concatenation of calculated   */
/* hexadecimal values.                                     */

data _null_ ;
```

```
   call execute('proc format fmtlib ; value pct');
   max=10000;
   maxloop=255;
   do i=1 to maxloop by 10;
      color='RGBA'||put(((maxloop)/(maxloop+i)*200),hex2.)
                  ||put(((maxloop)/(maxloop+i)*235),hex2.)
                  ||put(((maxloop)/(maxloop+i)*255),hex2.)||'95';
      from=max;
      to=(max+3000);
      max=max+3000;
      /* Create salary ranges of $3000.00 equal to the calculated RGBA color value.*/
      call execute(put(from,best.)||'-'||put(to,best.)||'='||quote(color));
   end;


   /* Create RGBA values for missing values and values outside the salary ranges.  */
   call execute('.="RGBAF7F5F0480" other="RGBAFF2A2A88"; run;');
 run;

data staff;
   infile datalines dlm='#';
   input Name $16. IdNumber $ Salary
         Site $ HireDate date7.;
   format hiredate date7.;
   datalines;
Capalleti, Jimmy# 2355# 21163# BR1# 30JAN09
Chen, Len#        5889# 20976# BR1# 18JUN06
Davis, Brad#      3878# 19571# BR2# 20MAR84
Leung, Brenda#    4409# 34321# BR2# 18SEP94
Martinez, Maria#  3985# 49056# US2# 10JAN93
Orfali, Philip#   0740# 50092# US2# 16FEB03
Patel, Mary#      2398# 35182# BR3# 02FEB90
Smith, Robert#    5162# 40100# BR5# 15APR66
Sorrell, Joseph#  4421# 38760# US1# 19JUN11
Zook, Carla#      7385# 22988# BR3# 18DEC10
;
run;
ods html close;
ods pdf file='outpdf.pdf';
proc print data=staff noobs label
           style(HEADER)={background=rgbac7eafe95  fontstyle=italic}
           style(DATA)={foreground=black};
   var name IdNumber ;
   var salary /style(DATA)={background=pct.};
     label IdNumber='Employee Number' salary='Salary in U.S. Dollars';
   format salary dollar7.;
   title 'Generated Colors for the Variable Salary';
run;
ods pdf close;
```

ログ 15.1 *RGBA カラーを使用した静的で異なる表の背景色*

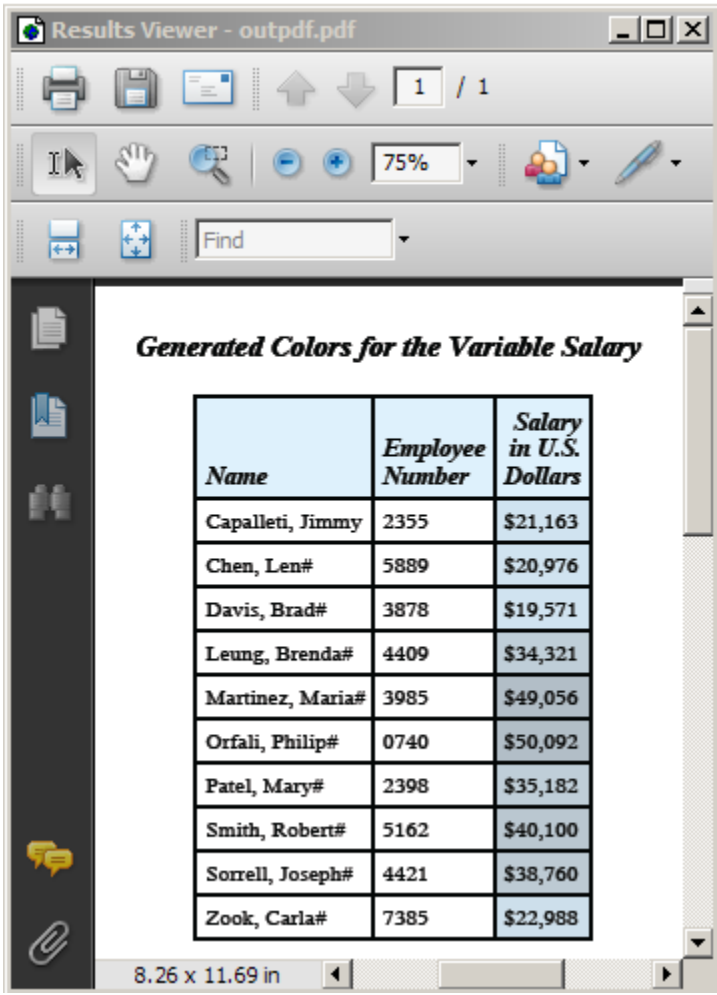
```
501  options nodate; 502 503  /* Create the PCT
format.                                 */ 504  /* The color variable is a
concatenation of calculated   */ 505  /* hexadecimal
values.                                    501 options nodate; 502 503 /* Create
the PCT format. */ 504 /* The color variable is a concatenation of calculated */
505 /* hexadecimal values. */ 506 507 data _null_ ; 508 call execute('proc
format fmtlib ; value pct'); 509 max=10000; 510 maxloop=255; 511 do i=1 to
maxloop by 10; 512 color='RGBA'||put(((maxloop)/(maxloop+i)*200),hex2.)||
put(((maxloop)/(maxloop+i)*235), 512! hex2.) 513 ||put(((maxloop)/(maxloop
+i)*255),hex2.)||'95'; 514 from=max; 515 to=(max+3000); 516 max=max+3000; 517
518 /* Create salary ranges of $3000.00 equal to the calculated RGBA color
value.*/ 519 call execute(put(from,best.)||'-'||put(to,best.)||'='||
quote(color)); 520 end; 521 522 /* Create RGBA values for missing values and
values outside the salary ranges. */ 523 call execute('.="RGBAF7F5F0480"
other="RGBAFF2A2A88"; run;'); 524 run; NOTE:513                ||put(((maxloop)/
(maxloop+i)*255),hex2.)||'95'; 514        from=max; 515        to=(max+3000);
516        max=max+3000; 517 518        /* Create salary ranges of $3000.00
equal to the calculated RGBA color value.*/519        call
execute(put(from,best.)||'-'||put(to,best.)||'='||quote(color)); 520     end;
521 522     /* Create RGBA values for missing values and values outside the
salary ranges.*/ 523     call execute('.="RGBAF7F5F0480" other="RGBAFF2A2A88";
run;'); 524   run; NOTE:DATA statement used (Total process time): real time 0.00
seconds cpu time 0.00 seconds
```

```
NOTE:CALL EXECUTE generated line.1   + proc format fmtlib ; 1
+                      value pct 2   +        10000-13000="RGBAC7EAFE95" 3
+        13000-16000="RGBABFE1F495" 4   +        16000-19000="RGBAB8D9EB95" 5
+        19000-22000="RGBAB2D1E395" 6   +        22000-25000="RGBAACCADB95" 7
+        25000-28000="RGBAA6C3D495" 8   +        28000-31000="RGBAA1BDCD95" 9
+        31000-34000="RGBA9CB7C795" 10  +        34000-37000="RGBA97B2C195" 11
+        37000-40000="RGBA93ADBB95" 12  +        40000-43000="RGBA8FA8B695" 13
+        43000-46000="RGBA8BA3B195" 14  +        46000-49000="RGBA879FAC95" 15
+        49000-52000="RGBA849BA895" 16  +        52000-55000="RGBA8097A495" 17
+        55000-58000="RGBA7D93A095" 18  +        58000-61000="RGBA7A909C95" 19
+        61000-64000="RGBA778C9895" 20  +        64000-67000="RGBA74899595" 21
+        67000-70000="RGBA72869195" 22  +        70000-73000="RGBA6F838E95" 23
+        73000-76000="RGBA6D808B95" 24  +        76000-79000="RGBA6B7D8895" 25
+        79000-82000="RGBA687B8595" 26  +        82000-85000="RGBA66788395" 27
+        85000-88000="RGBA64768095" 28  + .="RGBAF7F5F0480"
other="RGBAFF2A2A88"; NOTE:Format PCT has been output.28
+                                       run; NOTE:PROCEDURE FORMAT used (Total
process time): real time           0.03 seconds cpu time            0.01 seconds
525 526  data staff; 527     infile datalines dlm='#'; 528     input Name
$16.IdNumber $ Salary 529           Site $ HireDate date7.; 530     format
hiredate date7.; 531     datalines; NOTE:The data set Work.Staff has 10
observations and 5 variables.NOTE:DATA statement used (Total process time): real
time           0.01 seconds cpu time            0.01 seconds 542  ; 543  run; 544
```

```
545  /* Close the HTML destination and open the PDF destination.*/546  /* Format
the header background using an RGBA color.       */ 547  /* Use the PCT. format
to format the salary variable.      */ 548 549  ods html close; 550  ods pdf
file='outpdf.pdf'; NOTE:Writing ODS PDF output to DISK destination "c:\public
\mySASPrograms\outpdf.pdf", printer "PDF".551  proc print data=staff noobs label
552             style(HEADER)={background=rgbac7eafe95  fontstyle=italic}
553             style(DATA)={foreground=black}; 554     var name IdNumber ;
555     var salary /style(DATA)={background=pct.}; 556       label
IdNumber='Employee Number' salary='Salary in U.S. Dollars'; 557     format
salary dollar7.; 558     title 'Generated Colors for the Variable Salary'; 559
run; NOTE:There were 10 observations read from the data set
Work.Staff.NOTE:PROCEDURE PRINT used (Total process time): real time 0.03
seconds cpu time 0.03 seconds 560 ods pdf close; NOTE:ODS PDF printed 1 page to
c:\public\mySASPrograms\outpdf.pdf.561  ods html; NOTE:Writing HTML Body
file:sashtml7.htm
```

次に、フォーマットされた PDF 出力を示します。

アウトプット 15.1 *RGBA カラー値を使用してフォーマットされた PDF*



Name	Employee Number	Salary in U.S. Dollars
Capalleti, Jimmy	2355	$21,163
Chen, Len#	5889	$20,976
Davis, Brad#	3878	$19,571
Leung, Brenda#	4409	$34,321
Martinez, Maria#	3985	$49,056
Orfali, Philip#	0740	$50,092
Patel, Mary#	2398	$35,182
Smith, Robert#	5162	$40,100
Sorrell, Joseph#	4421	$38,760
Zook, Carla#	7385	$22,988

ユニバーサル印刷出力に非表示コメントを埋め込む

ファイルを表示や印刷するときには現れないコメントをユニバーサルプリンタ出力に埋め込みできます。そのコメントには 4,000 文字までのテキスト文字列が使用可能で、COLOPHON=システムオプションを使用して指定します。コメントをデジタル署名に使用したり、画像、ベクターグラフィックや PDF ファイルを識別したりする必要がある場合があります。テキストエディタやサードパーティのアプリケーションを使用して、ファイル内のテキスト文字列を表示できます。

この例では、COLOPHON=オプションを使用して SVG ドキュメントにテキストを追加します。

```
options printerpath=svg colophon='Colophon text: SVG SGPLOT for sashelp.class';
ods html close;
ods printer;
proc sgplot data=sashelp.class;
  reg x=height y=weight / CLM CLI;
run;
ods printer close;
ods html;
```

ここでは、SVG ドキュメントの 2 行目にコメントがあります。

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
<!-- Colophon text: SVG SGPLOT for sashelp.class -->
```

詳細については、“COLOPHON= System Option” (*SAS System Options: Reference*)を参照してください。

ウィンドウ環境を用いたユニバーサル印刷の設定

ユニバーサル印刷メニューの概要

SAS ユニバーサル印刷ウィンドウは、**ファイル**メニューからアクセスできます。

次のディスプレイは、ユニバーサル印刷の**ページ設定**、**印刷設定**、**印刷プレビュー**、**印刷**を含む**ファイル**メニューを示します。

図 15.5 *ユニバーサル印刷オプションを表示しているファイルメニュー*

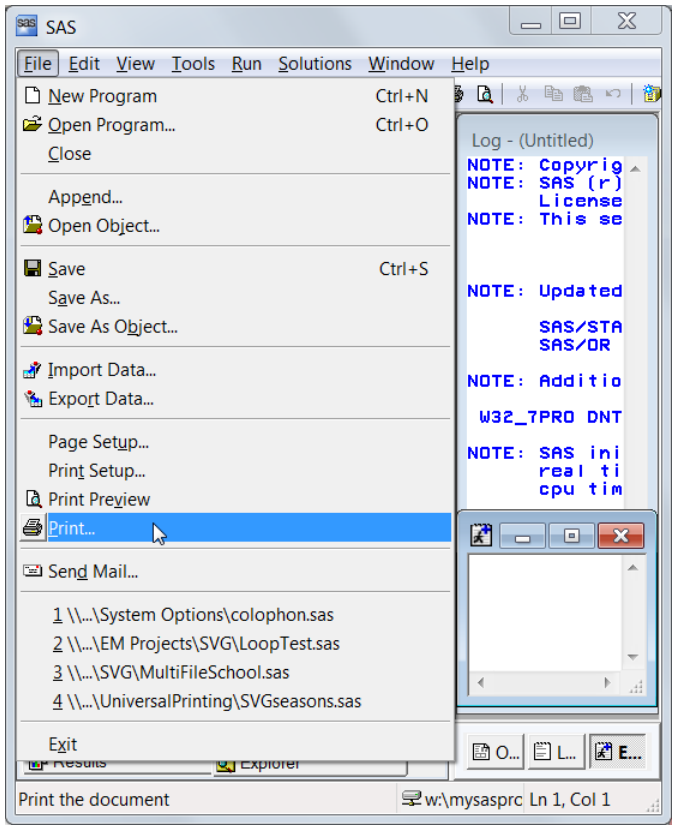


表 15.5 *ユニバーサル印刷ウィンドウを開くためのメニュー項目またはコマンド*

メニュー項目	対応するコマンド
ページ設定	DMPAGESETUP
印刷設定	DMPRINTSETUP

メニュー項目	対応するコマンド
印刷プレビュー	DMPRTPREVIEW
印刷	DMPRINT

Windows 固有
Windows 動作環境では、Windows 印刷ウィンドウがデフォルトとして使用されます。ユニバーサル印刷ユーザーインターフェイスにアクセスするには、UNIVERSALPRINT システムオプションを設定する必要があります。この場合、Windows で SAS を呼び出すための文字列に次のコード行を含めます。:

```
-uprint
```

UPRINT は、UNIVERSALPRINT システムオプションの別名です。

ユニバーサル印刷ウィンドウを開くには、コマンドをコマンドラインに入力するか、メニューバーのコマンドボックスに入力します。次の表では、ほとんどの共通タスクのコマンドを示します。

表 15.6 *ユニバーサル印刷ウィンドウを開くためのコマンド*

アクション	コマンド
現在のウィンドウを印刷します	DMPRINT
デフォルトプリンタを変更します	DMPRINT または DMPRINTSETUP
新しいプリンタまたはプレビューア定義を作成します	DMPRTCREATE PRINTER または DMPRTCREATE PREVIEWER
プリンタ定義を変更、追加、削除、テストします	DMPRINTSETUP
デフォルトプリンタプロパティシートを表示します	DMPRINTPROPS
ページプロパティシートを表示します	DMPAGESETUP
現在のウィンドウの印刷をプレビューします	DMPRTPREVIEW

プリンタの設定

印刷設定ウィンドウ

ファイル ⇨ **印刷設定**メニューを選択すると、**印刷設定**ウィンドウが表示されます。印刷設定ウィンドウでは、次のタスクを実行できます。

- デフォルトプリンタを変更する。
- 選択リストからプリンタを削除する。
- テストページを印刷する。
- **プリンタのプロパティ**ウィンドウを表示します。

- **新しいプリンタ**ウィザードを起動します。

または、DMPRINTSETUP コマンドを発行できます。

デフォルトプリンタの変更

現在以降の SAS セッションでデフォルトプリンタデバイスを変更するには、次の操作を実行します。

1. **ファイル** ⇨ **印刷設定**を選択します。**印刷設定**ウィンドウが表示されます。
2. **プリンタ**フィールドで、プリンタのリストから新規デフォルトデバイスを選択します。
3. **OK** をクリックします。

または、DMPRINTSETUP コマンドを発行できます。

選択リストからプリンタを削除する

選択リストからプリンタを削除するには、次の操作を実行します。

1. **ファイル** ⇨ **印刷設定**を選択します。**印刷設定**ウィンドウが表示されます。
2. **プリンタ**フィールドで、削除するプリンタをリストから選択します。
3. **削除**をクリックします。

注: システム管理者のみが、サイト用に定義しているプリンタを削除できます。システム管理者によって定義されたプリンタを選択した場合、**削除**ボタンは使用できません。

または、DMPRINTSETUP コマンドを発行できます。

新しいプリンタの定義

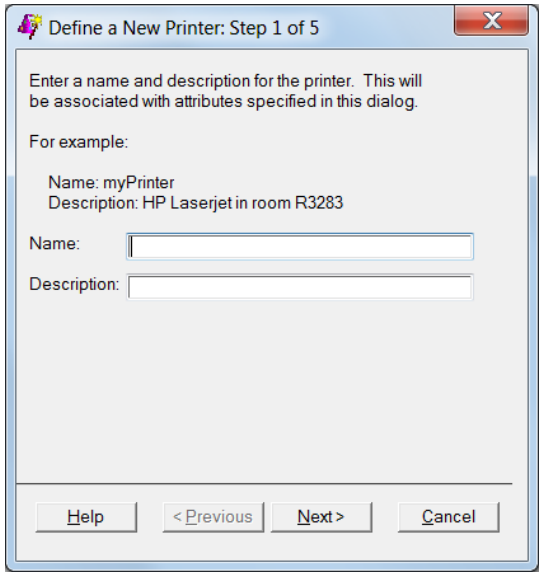
ユニバーサル印刷が定義済みプリンタを提供している間でも、**新しいプリンタの定義**ウィザードで独自のプリンタを追加できます。このウィザードでは、手順を追ってプリンタをインストールできます。

新しいプリンタウィザードを起動し、新しいプリンタを定義するには、次の操作を実行します。

1. **ファイル** ⇨ **印刷設定**を選択し、**新規作成**をクリックします。

   次のウィンドウが表示されます。

図 15.6 名前と説明を入力するためのプリンタ定義ウィンドウ



また、DMPRTCREATE PRINTER コマンドを発行するか、DMPRINTSETUP コマンドを発行し、**新規作成**をクリックします。

2. 新しいプリンタの名前と説明を入力します(最大 127 文字、バックスラッシュ文字なし、大文字と小文字の区別なし)。

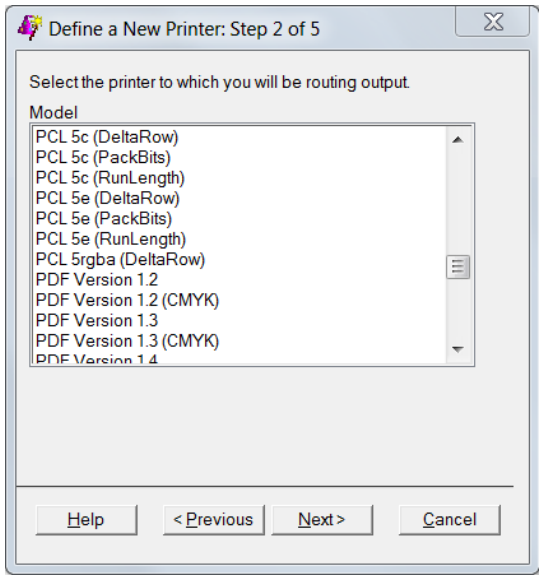
   プリンタ名は必須です。説明はオプションです。

3. **次へ**をクリックし、ウィザードのステップ 2 に進みます。

   プリンタモデルを選択します。正確なプリンタモデルがない場合は、そのプリンタと互換性のある汎用モデルを選択します。たとえば、HP LaserJet プリンタシリーズの場合、PCL5 (モノクロプリンタ)または PCL5c (カラープリンタ)を選択します。

   注: 汎用モデルの場合、特定のモデルよりも選択肢が少ない場合があります。

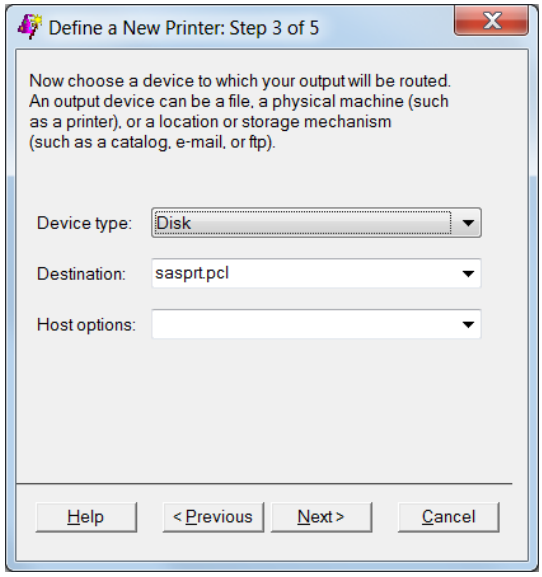
   図 15.7 プリンタモデルを選択するためのプリンタ定義ウィンドウ



4. **次へ**をクリックし、ウィザードのステップ 3 に進みます。

   次のウィンドウが表示されます。

図 15.8 出力デバイスを選択するためのプリンタ定義ウィンドウ



5. 印刷出力用の**デバイスタイプ**を選択します。番号付き項目の下のパラグラフにこの文を入れます。

   デバイスタイプの選択内容はホストによって異なります。

   カタログ、**ディスク**、**Ftp**、**ソケット**、または**パイプ**をデバイスタイプとして選択した場合、出力先を指定する必要があります。

   プリンタを選択した場合、一部の動作環境は出力を送信するためにホストオプションボックスを使用するので、出力先は不要の場合があります。

   注: デバイスタイプ、出力先、ホストオプションのオペレーティングシステムの例については、“デバイスタイプ、出力先、ホストオプションのフィールドのサンプル値” (266 ページ)にも記載されています。

6. ファイルの**出力先**を入力します。

   出力先は、デバイスタイプに応じた出力先です。たとえば、デバイスタイプが**ディスク**の場合、出力先は動作環境独自のファイル名です。一部のシステムデバイスタイプでは、出力先が空白なので、**ホストオプション**ボックスでターゲットの場所を指定できます。

7. 選択したデバイスのホスト固有のオプションを選択または入力します。

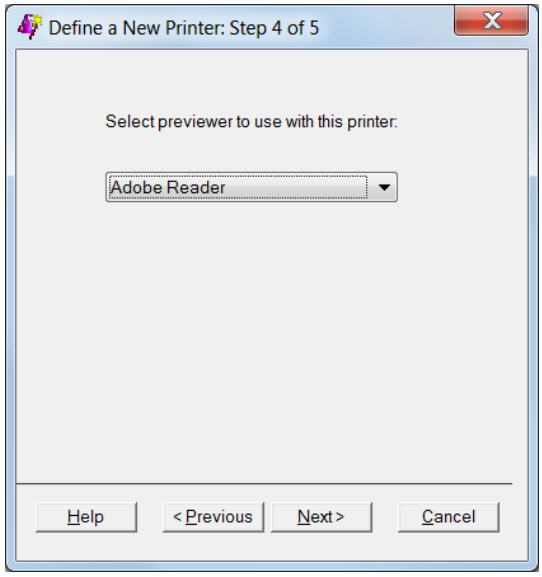
   このフィールドは、動作環境でオプションの場合もあります。ホストオプションのリストについては、動作環境の FILENAME ステートメント情報を参照してください。

   注: **出力先**リストと**ホストオプション**リストでは、REGISTRY プロシジャを使用して値を入力することもできます。ステップ 3 で**ヘルプ**ボタンをクリックすると、“出力先およびホストオプションリストの取り込み”トピックで詳細を参照できます。

8. **次へ**をクリックしてウィザードのステップ 4 に進み、インストールされている印刷プレビューアのリストから選択します。

   プレビューアが定義されていない場合は、ウィザードの次のステップに進みます。

図 15.9 プレビューアを選択するためのプリンタ定義ウィンドウ



プレビューア選択ボックスが表示されたら、このプリンタのプレビューアを選択します。プレビューアが不要な場合は、**なし**を選択するか、フィールドを空白のままにします。

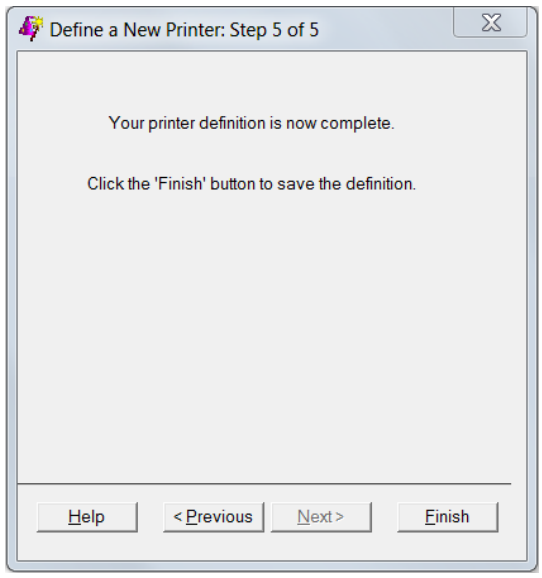
注: DMPRTCREATE PREVIEWER コマンドで、任意のプリンタにプレビューアを追加します。詳細については、“新しいプレビューアの定義” (253 ページ)を参照してください。

注: プリンタと印刷プレビューアは同じ言語を使用していなくてもかまいません。

9. **次へ**をクリックし、ウィザードのステップ 5 に進みます。

   次のウィンドウが表示されます。

図 15.10 処理を完了するためのプリンタ定義ウィンドウ



10. 情報を変更する場合は、**戻る**をクリックします。プリンタ定義を終了したら、**完了**をクリックします。

これで、デフォルトプリンタの設定は完了です。

印刷設定ウィンドウに戻ったら、**テストページ印刷**をクリックすると、デフォルトプリンタをテストできます。

注: また、PRTDEF プロシジャを使用してプリンタをプログラムで定義することもできます。詳細については、"PRTDEF プロシジャを用いたユニバーサルプリンタの管理" (261 ページ)を参照してください。

デフォルトプリンタのプリンタプロパティの設定

変更可能なプリンタプロパティには、次のものがあります。

- プリンタ名と説明
- プリンタ出力先デバイスとプロパティ
- プリンタのデフォルトフォント
- プリンタに関連付けられた変換テーブル、プリンタ解像度、印刷プレビューアなどの高度な機能

デフォルトプリンタのプリンタプロパティを変更するには、次の操作を実行します。

1. **ファイル** ⇨ **印刷設定**を選択し、**プロパティ**を選択します。

   プリンタのプロパティウィンドウが表示されます。

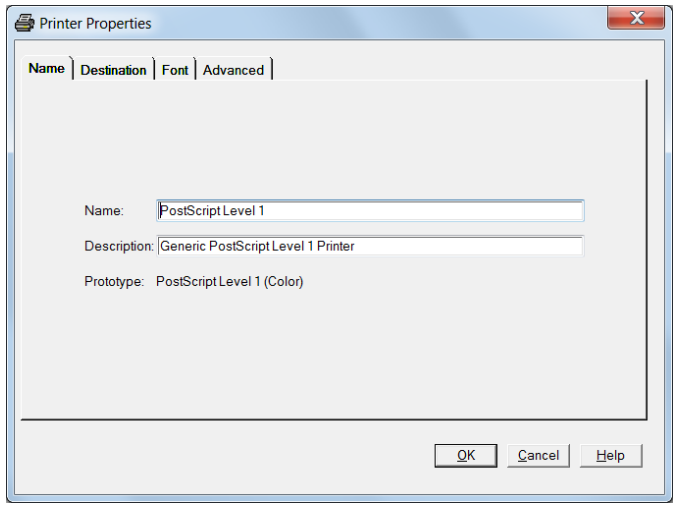
   または、DMPRTPROPS コマンドを発行できます。

2. **プリンタのプロパティ**ウィンドウで、変更する必要がある情報を含んでいるタブを選択します。

   - **名前**タブで、プリンタ名と説明を変更できます。

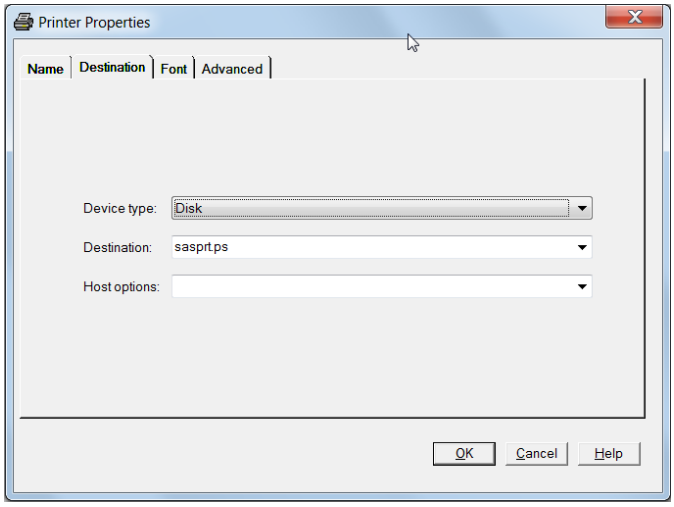
     注: プリンタ名では、大文字と小文字を区別しません。大文字と小文字の変更だけでは、プリンタ名の変更は失敗します。プリンタ名の大文字と小文字を変更するには、プリンタを削除し、新しい大文字と小文字の組み合わせで作成しなおします。または、プリンタの名前を変更し、変更を保存してから、目的の名前と大文字と小文字の組み合わせに名前を再び変更します。

   図 15.11 *名前タブを表示しているプリンタのプロパティウィンドウ*



   - **出力先**タブでは、プリンタのデバイスタイプ、出力先、ホストオプションを指定できます。例については、"デバイスタイプ、出力先、ホストオプションのフィールドのサンプル値" (266 ページ) を参照してください。

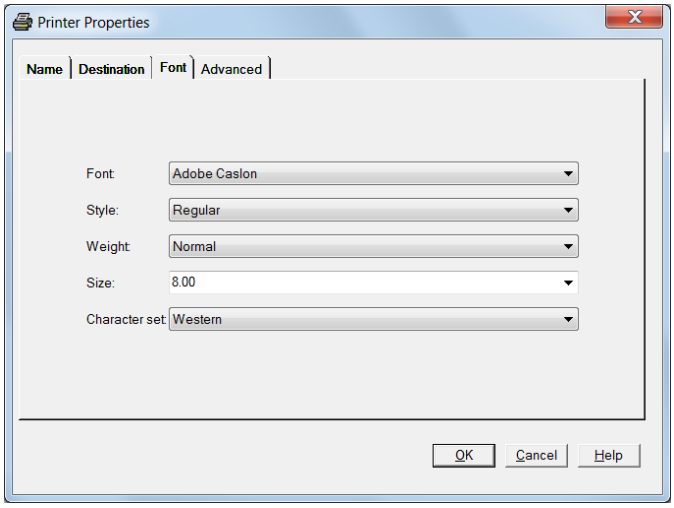
図 15.12 出力先タブを表示しているプリンタのプロパティウィンドウ



- **フォント**タブでは、使用可能なフォントオプションを制御します。ドロップダウンボックスで選択可能な内容は、プリンタによって異なります。フォントサイズはポイント単位で示されます。

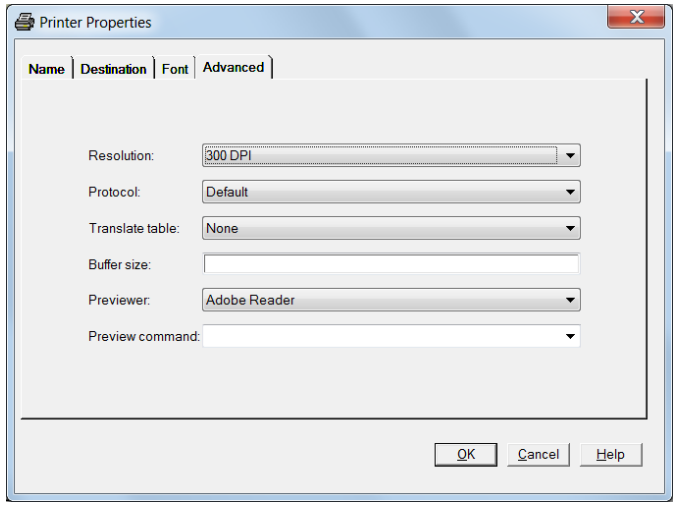
  注: このウィンドウでは、デフォルトフォントの属性を設定できます。通常、プロシジャ出力は、ODS スタイルで指定されたフォントまたはフォント属性を指定するプログラムステートメントで制御されます。

図 15.13 フォントタブを表示しているプリンタのプロパティウィンドウ



- **詳細**タブには、プリンタの解像度、プロトコル、変換テーブル、バッファサイズ、プレビューア、プレビューコマンドオプションが表示されます。ドロップダウンフィールドの情報は、プリンタによって異なります。

図 15.14 詳細タブを表示しているプリンタのプロパティウィンドウ



解像度
印刷された出力の解像度を dpi で指定します。

プロトコル
EBCDIC ホストメインフレームを ASCII デバイスに接続するプロトコルコンバータで処理可能な形式に出力を変換するメカニズムを提供します。プロトコルは、z/OS 動作環境では必須です。使用する場合は、リストされているプロトコルコンバータのいずれか 1 つを選択します。

変換テーブル
EBCDIC ホストと ASCII デバイスとの間のデータの変換を管理します。通常、ドライバはロケールに応じたテーブルを選択します。変換テーブルは、標準以外の変換が必要な場合にのみ指定します。

バッファサイズ
出力バッファまたはレコード長のサイズを制御します。バッファサイズを空白のまま残した場合、デフォルトサイズが使用されます。

プレビューア
印刷プレビューが要求されたときに使用するプレビューア定義を指定します。プレビューアボックスには、定義済みのプレビューアアプリケーションが表示されます。“新しいプレビューアの定義” (253 ページ)を参照してください。

プレビューコマンド
外部プリンタ言語ビューアを開く場合に使用するコマンドです。たとえば、Ghostview をプレビューアとして使用する場合は、`ghostview %s` と入力します。プレビューコマンドをプリンタ定義に入力すると、プリンタ定義はプレビューア定義になります。プレビューコマンドは有効なコマンドになっている必要があります。プレビュープロセスの一部としてコマンドを実行している場合、%s は、プレビューコマンドの入力を含む一時ファイルの名前に置き換えられます。

注: **プレビューア**フィールドと**プレビューコマンド**フィールドの内容は相互排他的で、両立しません。コマンドパスを**プレビューコマンド**フィールドに入力すると、**プレビューア**ボックスが淡色表示されます。

現在のセッション用のプリンタの指定方法

RINTERPATH=システムオプションを使用すると、現在の SAS セッションで使用するユニバーサルプリンタを指定できます。このプリンタの指定は、別の SAS セッションでは保持されません。デフォルトプリンタを設定するウィンドウ環境がない場合、バッチモー

ドでは PRINTERPATH=システムオプションが主に使用されます。このオプションでは、文字列を値として受け付けます。次にその例を示します。

```
options printerpath=myprinter;
options printerpath="Print PostScript to disk";
```

注: プリンタ名に空白が含まれている場合、それを引用符で囲む必要があります。

2 箇所から現在定義されているプリンタのリストが入手できます。

- **印刷設定**ウィンドウの**プリンタ**フィールドのプリンタのリスト。
- 次のコードをサブミットします。

```
proc qdevice out=printers;
   printer _all_;
run;

proc print data=printers;
   var name desc;
   where nametype contains "Printer";
run;
```

PRINTERPATH=システムオプションでファイル参照名を指定して、プリンタの出力先を無効にすることもできます。

```
options printerpath= (myprinter printout);
filename printout path;
```

ユニバーサル印刷での印刷

テストページ印刷

テストページを印刷するには、次の操作を実行します。

1. **ファイル** ⇨ **印刷設定**を選択し、**テストページ印刷**を選択すると、**印刷設定**ウィンドウが表示されます。
2. **プリンタ**リストビューからテストページを印刷するプリンタを選択します。
3. **テストページ印刷**をクリックします。

または、DMPRINTSETUP コマンドを発行できます。

アクティブな SAS ウィンドウの内容を印刷

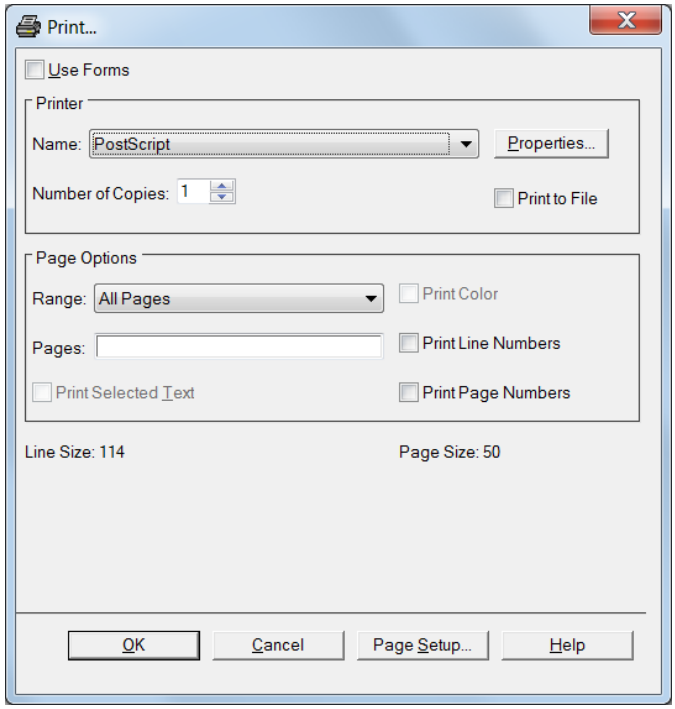
SAS でウィンドウの内容を印刷するには、次の操作を実行します。

1. ウィンドウ内をクリックしてアクティブにします。
2. **ファイル** ⇨ **印刷**を選択します。

   印刷ウィンドウが表示されます。印刷ウィンドウは、次に表示されるウィンドウとは違っている可能性があります。

   または、DMPRINT コマンドを発行できます。

図 15.15 印刷ウィンドウ



3. **フォームを使用する**チェックボックスが表示されている場合、ユニバーサル印刷を使用するにはオフにします。
4. 印刷グループボックスで、印刷定義の名前を選択します。
5. 必要な部数を入力します。
6. 印刷ジョブをファイルに保存する場合は、次の操作を実行します。
   a. **ファイルへ出力**を選択します。
   b. **OK** をクリックします。**ファイル選択**ウィンドウが表示されます。
   c. 既存のファイルを選択するか、新しいファイル名を入力します。

   注: 既存のファイルに印刷する場合、ファイルの内容は上書きされるか追加されます。これは、**置換**を選択するか、**追加**を選択するかによって異なります。EMF、GIF、PNG、SVG、TIFF ファイルの大半のビューアでは、追加されたファイルは表示されません。PDF プリンタで**追加**が選択されている場合、マージされた PDF ファイルは生成されません。
7. 追加印刷オプションを設定します。

   ページオプション領域のフィールドには、印刷する SAS ウィンドウの内容に従った選択内容が表示されます。デフォルトでは、SAS は選択されたウィンドウの内容全体を印刷します。

表 15.7 ページオプション

印刷する項目	実行内容
ウィンドウのテキストの選択行 *注:* z/OS では無効	印刷するテキストを選択し、**印刷**ウィンドウを開きます。**ページオプション**ボックスで、**選択した部分**ボックスを選択します。

印刷する項目	実行内容
ウィンドウに現在表示されているページ	**現在のページ**を選択します。
他の個別のページまたはページの範囲	**範囲**を選択し、**ページ**フィールドにページ番号を入力します。個別のページ番号またはページ範囲をカンマ(,)またはブランクで分けます。次の形式のいずれかでページ範囲を入力できます。 • n–m は、n - m の全ページを印刷します。 • –n は、1 - n ページの全ページを印刷します。 • n–は、n ページから最終ページまでの全ページを印刷します。
カラー	**カラーで印刷する**ボックスを選択します。
行番号	**行番号を印刷する**ボックスを選択します。
ページ番号	**ページ番号を印刷する**ボックスを選択します。
グラフ	DMPRINT コマンドを使用するか、**ファイル** ⇨ **印刷**を選択します。ユニバーサル印刷を使用するために **SAS/GRAPH ドライバを使用する**チェックボックスの選択が解除されていることを確認します。

8. 印刷するには、**OK** をクリックします。

プレビューアの操作

新しいプレビューアの定義

プレビューアを使用すると、印刷ジョブをプレビューできます。SAS は、デフォルトプレビューアアプリケーションを設定しません。印刷プレビュー機能を SAS で使用するには、ユーザーまたはシステム管理者は、最初にシステムのプレビューアを定義する必要があります。

z/OS 固有

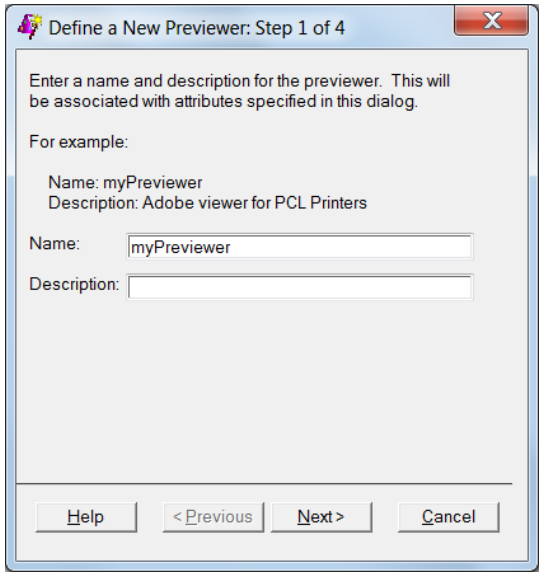
印刷プレビューアは z/OS でサポートされていません。

プレビューアは、**新しいプレビューア**ウィザードで定義できます。**新しいプレビューア**ウィザードを使用して新しい印刷プレビューアを定義するには、次の操作を行います。

1. DMPRTCREATE PREVIEWER コマンドを発行します。

   次のウィンドウが表示されます。

図 15.16 名前と説明を入力するためのプレビューア定義ウィンドウ

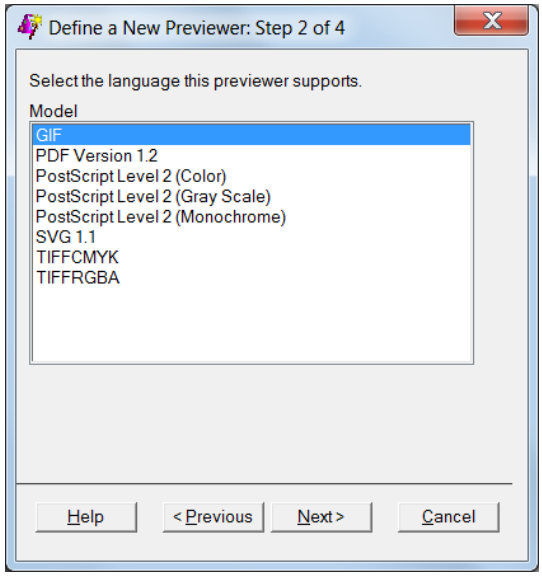


2. 新しいプレビューアの名前と説明を入力します(最大 127 文字、バックスラッシュなし、大文字と小文字の区別なし)。

   プレビューア名は必須です。説明はオプションです。

3. **次へ**をクリックし、ウィザードのステップ 2 に進みます。

図 15.17 プレビューア言語を入力するためのプレビューア定義ウィンドウ

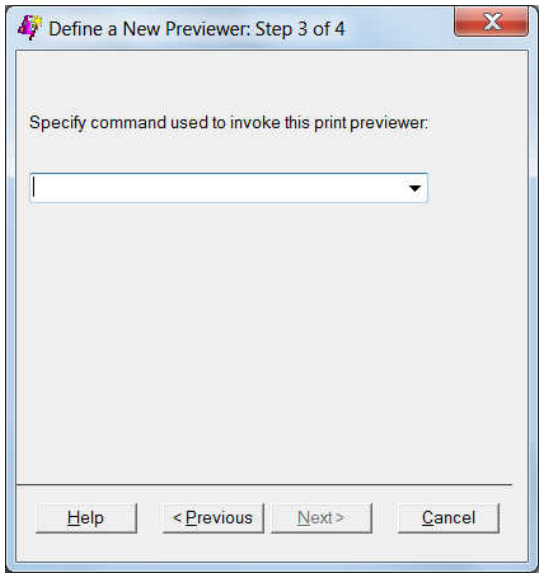


4. プレビューア定義に関連付けるプリンタモデルを選択します。

   モデル用に生成された PostScript、PCL、または PDF 言語は、外部ビューアパッケージがサポートしている言語である必要があります。最適の結果を得るには、PostScript Level 1 (カラー)または PCL 5 などの汎用モデルを選択します。

5. **次へ**をクリックし、ウィザードのステップ 3 に進みます。

図 15.18 プレビューアアプリケーションを開くコマンドを入力するためのプレビューア定義ウィンドウ



6. プレビューアアプリケーションを開くためのコマンドを入力し、その後に`%s` (通常は入力ファイル名が入ります)を追加します。

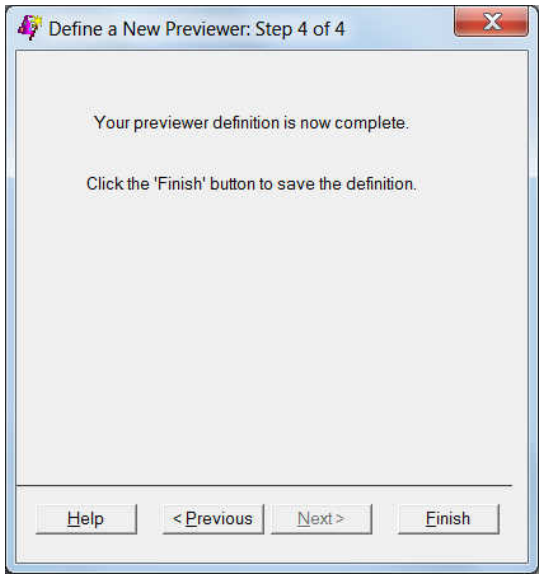
   たとえば、プレビューアを起動するためのコマンドが“ghostview”である場合、テキストフィールドに **`ghostview %s`** と入力します。

   印刷プレビューアプリケーションの呼び出しに使用するためのコマンドのリストをあらかじめ入力またはシードしておくこともできます。詳細については、“印刷プレビューアコマンドボックスの事前割り当て” (256 ページ)を参照してください。

   注: `%s` ディレクティブは、ビューアを起動するコマンドで必要な分だけ何度でも使用できます。ただし、起動コマンドは、マシンのコマンドパスにない場合、完全修飾されている必要があります。

7. **次へ**をクリックし、ウィザードのステップ 4 に進みます。

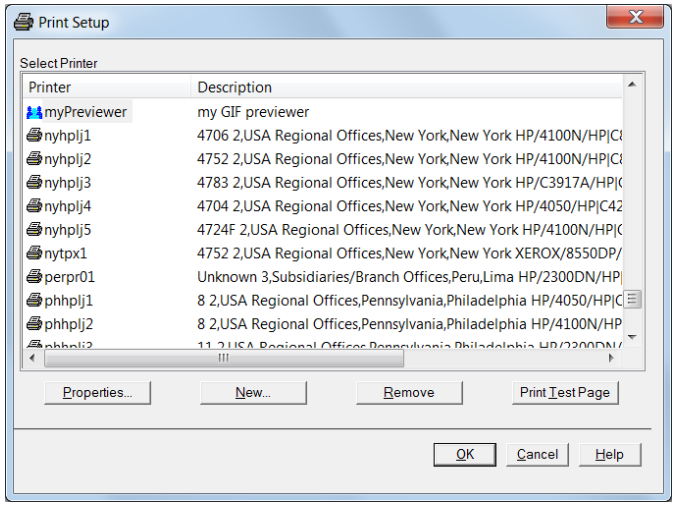
図 15.19 処理を完了するためのプレビューア定義ウィンドウ



8. 情報を修正する場合は、**戻る**をクリックします。デフォルトプレビューアの定義が完了したら、**完了**をクリックします。

新しく定義したプレビューアには、**印刷設定**ウィンドウにプレビューアアイコンが表示されます。

図 15.20 *新しいプレビューアを表示している印刷設定ウィンドウ*



このプレビューアプリケーションは、**印刷設定**ウィンドウの**テストページ印刷**ボタンでテストできます。

印刷プレビューアコマンドボックスの事前割り当て

印刷プレビューは、Ghostview、gv、Adobe Reader などの印刷プレビューアアプリケーションによってサポートされています。**プレビューア定義ウィザード**の**プレビューコマンド**ボックス(図 15.18 (255 ページ))および**プリンタのプロパティ**ウィンドウの**詳細**タブ(図 15.14 (250 ページ))に、コマンドのリストを事前に割り当てること(これをシードと呼びます)ができます。これらのコマンドは、サイトで使用可能な印刷プレビューアアプリケーションを呼び出すために使用します。ユーザーと管理者は、レジストリを手動で更新することも、プレビューアコマンドのリストを含むレジストリファイルを定義してインポートすることもできます。次に、レジストリファイルの例を示します。

```
[CORE\PRINTING\PREVIEW COMMANDS]
 "1"="/usr/local/gv %s"
 "2"="/usr/local/ghostview %s"
```

印刷ジョブのプレビュー

指定したプリンタ用に印刷ビューアがインストールされている場合、印刷プレビュー機能を使用できます。.印刷プレビューは、SAS のファイルメニューから常に選択できます。DMPRTPREVIEW コマンドを発行することもできます。

ページプロパティの設定

印刷出力の表示方法をカスタマイズするには、**ページ設定**ウィンドウを使用します。現在設定しているプリンタによっては、以降のステップに出てくるページ設定オプションの一部が使用できない場合があります。

印刷結果をカスタマイズするには、次の操作を実行します。

1. **ファイル** ⇨ **ページ設定**を選択します。

ページ設定ウィンドウが表示されます。

または、DMPAGESETUP コマンドを発行できます。

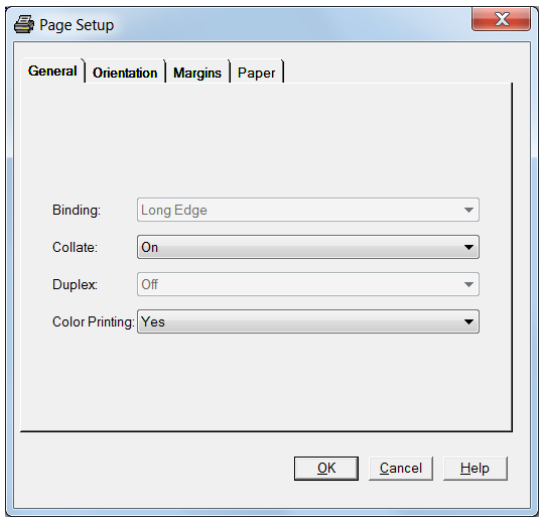
2. 印刷結果のさまざまな側面を制御するウィンドウを開くためのタブを選択します。

タブウィンドウの説明が後に続きます。

ページ設定ウィンドウは、**全般**、**印刷の向き**、**余白**、**用紙**という 4 つのタブから構成されます。

- **全般**タブでは、**とじしろ**、**部単位で印刷**、**両面印刷**、**カラー印刷**のオプションを変更できます。

図 15.21 *ページ設定ウィンドウの全般タブ*



とじしろ
両面印刷で使用するとじしろの縁(長辺側または短辺側)を指定します。これで、とじしろオプションが設定されます。

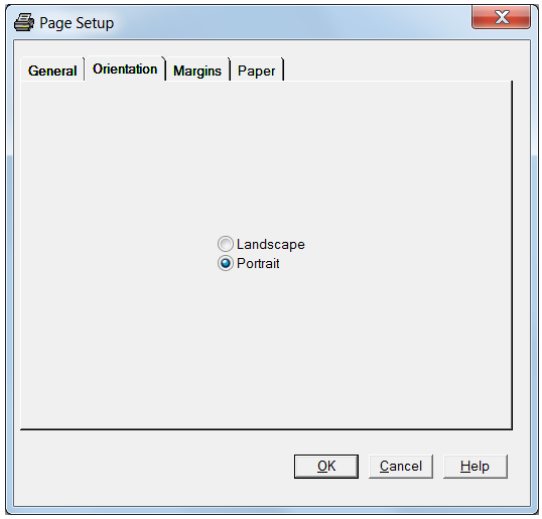
部単位で印刷
印刷結果を部単位で印刷するかどうかを指定します。これで、部単位で印刷オプションが設定されます。

両面印刷
片面に印刷するか両面に印刷するかを指定します。これで、両面印刷オプションが設定されます。

カラー印刷
印刷結果をカラーで印刷するかどうかを指定します。これで、COLORPRINTING オプションが設定されます。

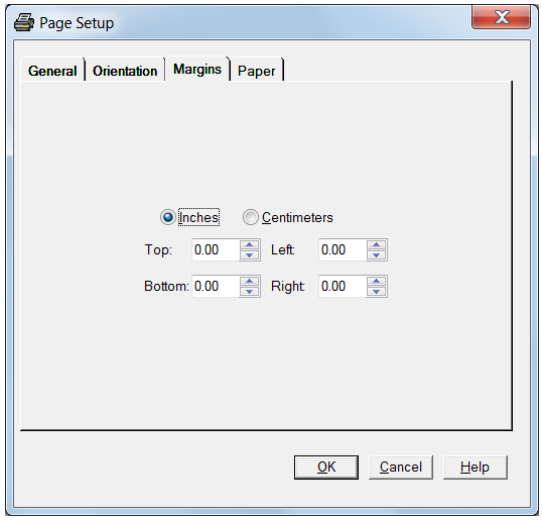
- **印刷の向き**タブでは、印刷結果のページの向きを変更できます。デフォルトは**縦**です。このタブでは、ORIENTATION オプションが設定されます。

図 15.22 ページ設定ウィンドウの印刷の向きタブ



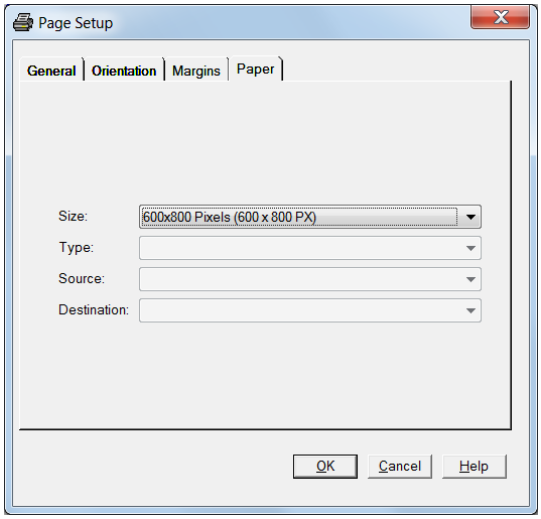
- **余白**タブでは、ページの上下左右の余白を変更できます。値の範囲は、使用しているプリンタのタイプによって異なります。このタブで指定した値によって、TOPMARGIN、BOTTOMMARGIN、LEFTMARGIN、RIGHTMARGIN オプションが設定されます。

図 15.23 ページ設定ウィンドウの余白タブ



- **用紙**タブでは、印刷に使用する用紙のサイズ、種類、トレイ、給紙先を指定します。

図 15.24 ページ設定ウィンドウの用紙タブ



サイズ
PAPERSIZE オプションを設定して、使用する用紙のサイズを指定します。用紙サイズには、レター、リーガル、A4 などがあります。

種類
使用する用紙の種類を指定します。選択内容には、普通紙、光沢紙、フィルムなどがあります。これで、PAPERTYPE オプションが設定されます。

トレイ
使用する用紙トレイを指定します。これで、PAPERSOURCE オプションが設定されます。

給紙先
印刷に使用するビンまたは用紙トレイを指定します。これで、PAPERDEST オプションが設定されます。

注: 用紙設定は SAS レジストリに格納されます。ページ設定は別の SAS セッションになっても有効ですが、デフォルトプリンタを変更すると、設定の一部が失われるか、変更される、または無効になる可能性があります。SAS セッション中にプリンタを変更する場合、**ページ設定**ウィンドウをチェックし、新しいデフォルトプリンタの設定がすべて有効であることを確認します。

ユニバーサル印刷を制御するシステムオプション

次のシステムオプションはユニバーサル印刷を制御します。

表 15.8 ユニバーサル印刷を制御するシステムオプション

システムオプション	説明
BINDING=	プリンタのとじしろを指定します。
BOTTOMMARGIN=	印刷する際のページ下部の余白サイズを指定します。

システムオプション	説明
COLLATE	プリンタで、複数のコピーを部単位で印刷するように指定します。
COLORPRINTING	サポートされている場合はカラー印刷を指定します。
COPIES=	印刷時の部数を指定します。
DUPLEX	サポートされている場合は両面印刷を指定します。
LEFTMARGIN=	ページ左側の余白サイズを指定します。
ORIENTATION=	ドキュメント全体またはドキュメント内の個別ページの用紙の向き(縦、横、縦逆、横逆)を指定します。
PAPERDEST=	印刷結果を出力するビンまたは用紙トレイを指定します。
PAPERSIZE=	印刷時に使用する用紙サイズを指定します。
PAPERSOURCE=	印刷に使用する用紙トレイを指定します。
PAPERTYPE=	印刷に使用する用紙の種類を指定します。
PRINTERPATH=	ユニバーサル印刷印刷ジョブで使用するプリンタを指定します。
RIGHTMARGIN=	ページ右側の余白サイズを指定します。
SYSPRINTFONT=	印刷時に使用するデフォルトフォントを指定します。
TOPMARGIN=	ページ上部の余白サイズを指定します。

注: PRINTERPATH=システムオプションは、使用するプリンタを指定します。

- PRINTERPATH=システムオプションがブランクの場合は、デフォルトプリンタを使用します。
- PRINTERPATH=システムオプションがブランクでない場合は、ユニバーサル印刷を使用します。

注: Windows 環境では、デフォルトプリンタは現在の Windows システムプリンタまたは SYSPRINT システムオプションで指定したプリンタです。そのため、ユニバーサル印刷は使用されません。

PRTDEF プロシジャを用いたユニバーサルプリンタの管理

PRTDEF プロシジャの使用について

プリンタ定義の作成には、PRTDEF プロシジャを使用します。個々の SAS ユーザーに対しても、サイトの全 SAS ユーザーに対しても作成できます。PRTDEF プロシジャを使用すると、ユニバーサル印刷ウィンドウで実行可能なプリンタ管理アクティビティの多くを実行できます。PRTDEF プロシジャは、どの実行モードでも使用できますが、ユニバーサル印刷ウィンドウを使用できないバッチモードで SAS を使用する場合に特に役立ちます。

PRTDEF プロシジャで 1 つ以上のプリンタを定義または変更するには、プリンタ属性に対応する変数を格納する SAS データセットを最初に作成します。どのプリンタの出力先でも、次の 4 つの変数を指定する必要があります。

DEST
プリンタの出力先を指定します。

DEVICE
デバイス名を指定します。

MODEL
プリンタのプロトタイプの名前を指定します。プリンタプロトタイプのリストについては、SAS レジストリを開き、\CORE\PRINTING\PROTOTYPES キーにアクセスします。

NAME
プリンタの名前を指定します。

オプション変数のリストについては、“Input Data Set: PRTDEF Procedure” (*Base SAS Procedures Guide*)を参照してください。PRTDEF プロシジャは、データセットを読み取り、変数属性を SAS レジストリの 1 つ以上のプリンタ定義に変換します。

プリンタ定義データセットを作成したら、PRTDEF プロシジャを実行してプリンタを作成します。

システム管理者または Sashelp ライブラリに対する書き込み権限を持つユーザーのみが、PRTDEF プロシジャを使用してサイトのすべての SAS ユーザーに対するプリンタ定義を作成できます。個々のユーザーは、Sasuser ライブラリに対する書き込み権限を持っており、PRTDEF プロシジャを使用して独自のプリンタを作成できます。ただし、プリンタ定義は Sasuser ライブラリに格納され、Sasuser ライブラリが削除されると失われます。個々のユーザーによって作成されたプリンタ定義は、プリンタ定義が格納されているディレクトリが Sasuser ライブラリと指定されている場合にのみ有効です。Sasuser ライブラリの割り当ての詳細については、“SASUSER= System Option” (*SAS System Options: Reference*)を参照してください。

詳細については、“PRTDEF” (*Base SAS Procedures Guide*)を参照してください。

PRTDEF プロシジャを用いた新しいプリンタとプレビューアの作成例

はじめに

次に、PRTDEF プロシジャを使用して新しいプリンタを定義する方法と、インストールされたプリンタとプレビューアを管理する方法の例を示します。

PRTDEF プロシジャを含んでいるプログラムステートメントが正常に実行されると、定義済みのプリンタまたはプレビューアが**印刷設定**ウィンドウに表示されます。使用可能なすべてのプリンタとプレビューアは**プリンタ名**リストに表示されます。プリンタ定義は、**レジストリエディタ**ウィンドウの CORE\PRINTING\PRINTERS でも参照できます。

複数のプリンタを定義するデータセットの作成

プリンタを定義する PRTDEF プロシジャで使用するデータセットを作成する際には、名前、モデル、デバイス、出力先の変数を指定する必要があります。

使用可能なオプション変数の名前については、*Base SAS Procedures Guide* の "PRTDEF" (*Base SAS Procedures Guide*)を参照してください。

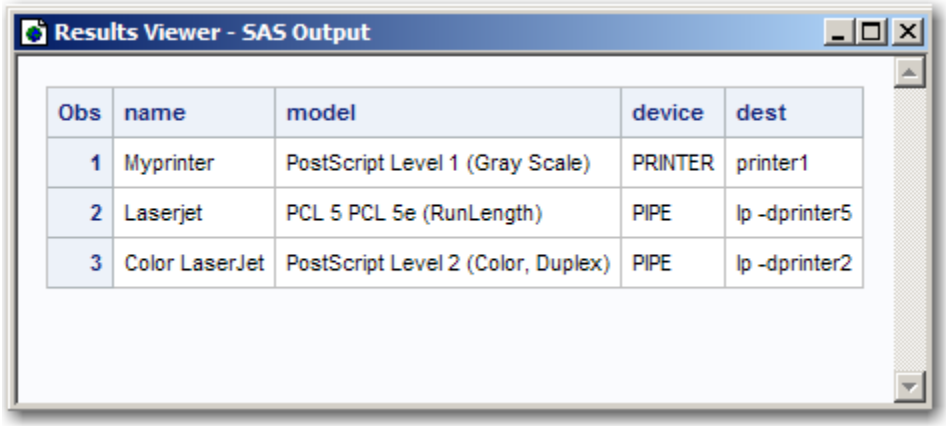
次のコードでは、PRTDEF プロシジャで使用するデータセットを作成します。

```
data printers;
   input name $15. model $35. device $8. dest $14.;
   datalines;
Myprinter      PostScript Level 1 (Gray Scale)     PRINTER printer1
Laserjet       PCL 5 PCL 5e (RunLength)           PIPE    lp -dprinter5
Color LaserJet PostScript Level 2 (Color, Duplex) PIPE    lp -dprinter2
;
run;

proc print data=printers;
run;
```

次のように出力されます。

アウトプット 15.2 *プリンタデータセット*



Obs	name	model	device	dest
1	Myprinter	PostScript Level 1 (Gray Scale)	PRINTER	printer1
2	Laserjet	PCL 5 PCL 5e (RunLength)	PIPE	lp -dprinter5
3	Color LaserJet	PostScript Level 2 (Color, Duplex)	PIPE	lp -dprinter2

変数を格納しているデータセットを作成したら、PRTDEF プロシジャを実行します。PRTDEF プロシジャは、SAS レジストリに該当するエントリを作成することで、データセットで指定されたプリンタを作成します。

```
proc prtdef data=printers usesashelp replace;
run;
```

USESASHELP オプションは、全ユーザーが使用できるように、Sashelp ライブラリにプリンタ定義を配置するよう指定します。USESASHELP オプションを指定しなかった場合、現在の Sasuser ライブラリにプリンタ定義が配置されます。プリンタ定義を使用できるのはローカルユーザーのみです。定義されているプリンタは、ローカルの Sasuser ディレクトリにおいてのみ使用可能です。ただし、USESASHELP オプションを使用するには、Sashelp ライブラリへの書き込み権限が必要です。

REPLACE オプションは、既存のプリンタ定義の変更がデフォルト操作になるように指定します。既存のプリンタ名は、プリンタ属性データセットの情報に基づいて変更されます。存在しないプリンタ名は追加されます。

複数のユーザー向けのプリンタの作成

次の例では、プレビューアプリケーションとして Ghostview を使用し、Sashelp ライブラリにプリンタ定義を格納するように指定した Tektronix Phaser 780 プリンタ定義を作成します。下部の余白は 2 cm、フォントサイズは 14 ポイント、用紙サイズは ISO A4 に設定されています。

```
data tek780;
   name = "Tek780";
   desc = "Test Lab Phaser 780P";
   model = "Tek Phaser 780 Plus";
   device = "PRINTER";
   dest = "testlab3";
   preview = "Ghostview";
   units = "cm";
   bottom = 2;
   fontsize = 14;
   papersiz = "ISO A4";
run;

proc prtdef data=tek780 usesashelp;
run;
```

注: このプリンタの出力結果をプレビューするには、Ghostview プリンタ定義を作成する必要があります。これを行うには、プレビュー定義ウィザード(図 15.14 (250 ページ))で、**プリンタのプロパティ**ウィンドウの**詳細**タブ(図 15.18 (255 ページ))を使用するか、または PRTDEF プロシジャを使用します。

PRTDEF プロシジャを使用した Ghostview プリンタ定義は次のようになります。

```
data gsview;
name = "Ghostview";
desc = "Print Preview with Ghostview";
model= "Tek Phaser 780 Plus";
viewer = 'gv %s';
device = "dummy";
dest = " ";

proc prtdef data=gsview list replace usesashelp;
run;
```

PROC PRTDEF ステートメントの LIST オプションは、プリンタ定義をログに書き込むように指定します。

注: プレビュー定義ウィザード(図 15.14 (250 ページ))または**プリンタのプロパティ**ウィンドウの**詳細**タブ(図 15.18 (255 ページ))のいずれかでプレビューコマンドを指定する必要があります。プレビューコマンドの例を次に示します。**`ghostview -bg white -fg black -magstep -2 -nolabel %s`**

印刷プレビューアの詳細については、“PostScript プレビューの定義の作成” (264 ページ)を参照してください。

プリンタの追加、変更、削除

この例では、Printers データセットを使用してプリンタ定義を追加、変更、削除します。これ以外にプリンタの定義に使用できる変数については、“PRTDEF” (*Base SAS Procedures Guide*)を参照してください。次のリストでは、例で使用する変数を示します。

- MODEL 変数には、このプリンタの定義時に使用するプリンタのプロトタイプを指定します。
- DEVICE 変数には、出力をプリンタに送信する際に使用する I/O デバイスの種類を指定します。
- DEST 変数には、プリンタの出力先を指定します。
- OPCODE 変数には、プリンタ定義で実行するアクション(追加、削除、変更)を指定します。
- 最初の Add 操作では、Color PostScript の新しいプリンタ定義がレジストリに作成され、2 番目の Add 操作では ColorPS の新しいプリンタ定義がレジストリに作成されます。
- Mod 操作では、レジストリにある LaserJet 5 の既存のプリンタ定義が変更されます。
- Del 操作では、“Gray PostScript”および“test”というプリンタのプリンタ定義がレジストリから削除されます。

次の例では、Sashelp ライブラリにプリンタ定義を作成します。定義は Sashelp にあるので、定義はすべてのユーザーで使用できるようになります。Sashelp ライブラリに書き込むには、特殊なシステム管理権限が必要です。個々のユーザーは、Sasuser ライブラリをかわりに指定することで個人のプリンタ定義を作成できます。

```
data printers;
   infile datalines dlm='#';
   length name $ 80
   model $ 80
   device $ 8
   dest $ 80
   opcode $ 3;
   input opcode $ name $ model $ device $ dest $ ;
datalines;
add#  Color PostScript F2#  PostScript Level 2 (Color)#      DISK#  sasprt.ps
mod#  LaserJet 5#           PCL 5c (DeltaRow)#               DISK#  sasprt.pcl
del#  Gray PostScript#      PostScript Level 2(Gray Scale)#  DISK#  sasprt.ps
del#  test#                 PostScript Level 2 (Color)#      DISK#  sasprt.ps
add#  ColorPS#              PostScript Level 2 (Color)#      DISK#  sasprt.ps
;

proc prtdef data=printers list;
run;
```

注: エンドユーザーは管理者が定義したプリンタの新しい属性を変更して Sashelp ライブラリに保存する場合、プリンタは Sasuser ライブラリのユーザー定義プリンタになります。ユーザーが指定した値は、管理者が設定した値を上書きします。ユーザー定義のプリンタ定義が削除された場合、管理者定義のプリンタが再び表示されます。

PostScript プレビューの定義の作成

次の例では、Adobe Acrobat Reader と Ghostview の両方の形式で PDF 出力をプレビューするために、Adobe Acrobat Reader 印刷プレビューアと Ghostview 印刷プレビュ

ーアを作成する方法を示します。データセット内の変数は、印刷プレビューア定義を SAS レジストリに生成するために PRTDEF プロシジャで使用する値を持ちます。

- NAME 変数には、プリンタ定義データレコードの残りの属性に関連付けられたプリンタ名を指定します。
- DESC 変数は、プリンタの説明を指定します。
- MODEL 変数には、このプリンタの定義時に使用するプリンタのプロトタイプを指定します。
- VIEWER 変数には、印刷プレビューのホストシステムコマンドを指定します。

  注: **ghostview %s** コマンドは、マシンのコマンドパスにない場合、完全修飾されている必要があります。

  注: プレビュー定義ウィザード(図 15.14 (250 ページ))または**プリンタのプロパティ**ウィンドウの**詳細**タブ(図 15.18 (255 ページ))のいずれかでプレビューコマンドを指定する必要があります。プレビューコマンドの例を次に示します。**ghostview -bg white -fg black -magstep -2 -nolabel %s** および **c:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe' %s.pdf**
- DEVICE 変数は、常に DUMMY にします。
- 出力が返されないようにするには、DEST をブランクにします。

次のプログラムでは、Adobe Acrobat Reader を使用するための印刷プレビューア定義を作成します。

```
data adobeR;
   name = "myAdobeReader";
   desc = "Adobe Reader Print Preview";
   model= "PDF Version 1.2";
   viewer = "'c:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe' %s.pdf";
   device = "dummy";
   dest = " ";
run;
proc prtdef data=adobeR list replace;
run;
```

次のプログラムでは、Ghostview を使用するための印刷プレビューア定義を作成します。

```
data gsview;
   name = "MyGhostview";
   desc = "Print Preview with Ghostview";
   model= "PostScript Level 2 (Color)";
   viewer = 'ghostview %s';
   device = "dummy";
   dest = " ";
run;
proc prtdef data=gsview list replace;
run;
```

プリンタ定義のエクスポートとバックアップ

PRTEXP プロシジャを使用すると、PRTDEF プロシジャで格納可能な SAS データセットとしてプリンタ定義をバックアップできます。

PRTEXP プロシジャの構文は次のとおりです。

```
PROC PRTEXP <USESASHELP> <OUT=dataset>
```

```
   <SELECT | EXCLUDE> printer_1 printer_2 ... printer_n;
```

次の例では、PRTEXP プロシジャを使用して 4 つのプリンタ定義(PDF、postscript、PCL5、PCL5c)をバックアップする方法を示します。

```
proc prtexp out=printers;
   select PDF postscript PCL5 PCL5c;
run;
```

詳細については、“PRTEXP” (*Base SAS Procedures Guide*)を参照してください。

デバイスタイプ、出力先、ホストオプションのフィールドのサンプル値

次のリストには、デバイスタイプ、出力先、ホストオプションのプリンタ値の例を示します。これらの値は互いに依存しており、値は動作環境によって異なるので、複数の例が示されています。プリンタをインストールする際、または出力先を変更する際に、このリストを参照できます。

- デバイスタイプ:プリンタ
  - z/OS
    - デバイスタイプ:プリンタ
    - 出力先:(空白のままにする)
    - ホストオプション: sysout=*class-value* dest=*printer-name*
  - UNIX および Windows
    - デバイスタイプ:プリンタ
    - 出力先:*プリンタ名*
    - ホストオプション:(空白のままにする)
- デバイスタイプ:パイプ

  注: 出力を UNIX ホストの lp 定義のプリンタキューに送信するサンプルコマンドは *lp -ddest* です。
  - UNIX
    - デバイスタイプ:パイプ
    - 出力先:*コマンド*
    - ホストオプション:(空白のままにする)
- デバイスタイプ:FTP

  注: ノード名の例は、*pepper.unx* です。
  - z/OS
    - デバイスタイプ:FTP
    - 出力先:ftp.out
    - ホストオプション: host=*'nodename'* recfm=vb prompt
    - デバイスタイプ:プリンタ
    - 出力先:*プリンタ名*
    - ホストオプション:(空白のままにする)
  - Windows
    - デバイスタイプ:FTP

    - 出力先:ftp.out
    - ホストオプション: host='*nodename*' prompt
  - UNIX
    - デバイスタイプ:FTP
    - 出力先: filename.ext
    - ホストオプション: host='*nodename*' prompt
- デバイスタイプ:ソケット

  注: lp 出力先キューの例は、*lp286nc0.prt:9100* です。
  - UNIX
    - デバイスタイプ:ソケット
    - 出力先:*destination-queue*
    - ホストオプション:(空白のままにする)

フォーム印刷

フォーム印刷の概要

ユニバーサル印刷を導入する前に、SAS では、フォームという印刷ジョブ用ユーティリティを提供していました。フォームは、印刷ジョブのページの行サイズや余白情報などを制御できる標準テンプレートです。ユニバーサル印刷の方が使いやすく、フォーム印刷よりも多くの機能を備えていますが、SAS では現在でもフォームをサポートしています。

フォームによる印刷は、**印刷**ウィンドウでも可能です。フォーム印刷モードに切り替えるには、**ファイル** ⇨ **印刷**を選択し、**フォームを使用する**を選択します。

注: フォーム印刷は、バッチモードでは使用できません。

フォームの作成または編集

フォームで印刷する必要があるレガシーレポートが組織にある場合、**FORM** ウィンドウを使用してフォームを作成または編集する必要がある可能性もあります。ユニバーサル印刷の方が機能が多く、印刷には推奨できる方法ですが、SAS ではまだフォームを作成または編集できます。

フォームを作成または編集するには、次の形式の FSFORM コマンドを入力します。

FSFORM<*catalog-name.*>*form-name*

カタログ名を指定しなかった場合、SASUSER.PROFILE カタログが使用されます。指定したフォーム名が存在しない場合、新しいフォームが作成されます。

新しいフォームを作成する場合、Printer Selection フレームが表示されます。既存のフォームを編集している場合、Text Body and Margin Information フレームが表示されます。

FORMS フレーム間を移動するには、次の操作を実行します。

- 次のフレームにスクロールするには NEXTSCR コマンドを使用し、前のフレームにスクロールするには PREVSCR コマンドを使用します。

- 等号(=)と移動先のフレームの番号を入力します。たとえば、`=1` には Text Body and Margin Information フレームが表示され、`=2` には Carriage Control Information フレームが表示されます。
- **ツール**メニューからフレームの名前を選択します。
- **ツール**メニューから**次のスクリーン**または**前のスクリーン**を選択します。

TAB キーでフレームのフィールド間を移動できます。

フォームの定義または編集を完了したら、END コマンドを発行して変更を保存し、**FORM** ウィンドウを終了します。

注: 印刷ウィンドウで**フォームを使用する**チェックボックスを選択してフォームを有効にすると、非グラフィックウィンドウを印刷するためにユニバーサル印刷が無効になります。

動作環境の情報
フォームによる印刷の詳細については、各動作環境に対応するドキュメントを参照してください。

ユニバーサルプリンタと SAS/GRAPH デバイスでのフォントの使用

フォントの表示

ユニバーサル印刷では、FreeType ライブラリを使用することにより、SAS 出力でのフォントの生成と表示を行います。FreeType ライブラリは TrueType フォントと Type1 フォントをサポートしています。SAS におけるこれらのフォントの表示は、FreeType ライブラリか、またはホストのフォント表示機能を通じて行われます。

SAS でサポートする動作環境のすべてでフォントを表示できるので、FreeType ライブラリを使用する出力方法の方が適しています。推奨されるユニバーサルプリンタと SAS/GRAPH デバイスを次に示します。これらは FreeType ライブラリを使用して [1] フォントを表示します。

注: ユニバーサル印刷および SAS/GRAPH は、2 バイト Type1 フォントをサポートしていません。

[1] FreeType ライブラリは、テキストの測定とフォントの表示という SAS の 2 つの操作を実行するために使用されます。FreeType ライブラリは、指定された出力デバイスに応じて、これらの操作の一方または両方を実行します。

表 15.9 フォントの表示に FreeType ライブラリを使用するデバイス

出力方法	デバイス
SAS/GRAPH デバイス	GIF、GIFANIM、HTML、WEBFRAME
	TIFFB、TIFFP、TIFFG3、TIFFB300、TIFFP300
	JPEG
	PCL5、PCL5C、PCL5E
	PDF、PDFC、PDFA
	PNG、PNGT、PNG300
	PSL、PSCOLOR、PSLEPSF、PSLEPSFC
	SVG、SVGT、SVGVIEW、SVGANIM、SVGZ*
	SASEMF、SASWMF**
	SASPRTC、SASPRTG、SASPRTM プリンタインターフェイスデバイス
ODS プリントとユニバーサル印刷	EMF、EMFDUAL**
	GIF
	PCL5、PCL5C、PCL5E
	PDF、PDFA
	PNG、PNGT、PNG300
	PostScript
	SVG、SVGT、SVGZ、SVGView、SVGnotip
	SVGANIM
ODS RTF	PNG, SASEMF *, EMF
ODS HTML	PNG、PNGT、PNG300、GIF、JPEG、SVG、SVGT *

* NOFONTRENDERING オプションを設定する場合、デバイスドライバは FreeType ライブラリをテキストの測定にのみ使用します。

** これらのデバイスは、テキストの測定にのみ FreeType ライブラリを使用します。最終的なフォントの表示は、システムによってインストールされたフォントを使用して出力を表示する Microsoft Word などのアプリケーションによって実行されます。

フォント埋め込みを使用すると、出力の作成で使用したフォントがその出力に含まれるので、本来のレイアウトで表示または印刷できるようになります。FONTEMBEDDING システムオプションの詳細については、“FONTEMBEDDING System Option” (*SAS System Options: Reference*)を参照してください。

注: すべてのブラウザがフォント埋め込みをサポートしているわけではありません。

次の表に、ホストのフォント表示のみを使用するデバイスと出力方法を示します。

表 15.10 *ホストのフォント表示のみを使用するデバイス*

出力方法	デバイス
SAS/GRAPH デバイス	ACTIVEX、ACTXIMG、JAVA、JAVAIMG *注:* これらはクライアントデバイスです。
	BMP
	EMF、WMF
	ZGIF

表 15.11 *FreeType フォント表示またはホストのフォント表示を使用するデバイス*

出力方法	デバイス	デフォルトで使用
SAS/GRAPH デバイス	ZGIF、ZPNG	ホストのフォント表示
	HTML、WEBFRAME	FreeType フォント表示
	TIFFB、TIFFP、TIFFB300、TIFFP300、	FreeType フォント表示
	JPEG	FreeType フォント表示

注: **OPTIONS FONTRENDERING= FREETYPE_POINTS | HOST_PIXELS** を設定すると、FreeType ライブラリおよびホスト表示をサポートするデバイスの表示方法を変更できます。

UNIX 固有

UNIX 動作環境でホスト表示を使用するデバイスの場合、使用中の X サーバー上に TrueType フォントをインストールする必要があります。これは通常、DISPLAY 環境変数で指定します。詳細については、*Configuration Guide for SAS 9.4 Foundation for UNIX Environments* を参照してください。

ODS スタイルと TrueType フォント

デフォルトでは、多くの SAS/GRAPH デバイスドライバとすべてのユニバーサルプリンタは、ODS スタイルを使用して出力を生成します。また、これらの ODS スタイルは TrueType フォントを使用します。スタイルが指定されていない場合は、デフォルトのスタイルが使用されます。グラフの表示を、SAS 9.2 より以前に生成されたグラフと互換性があるものにする場合は GSTYLE システムオプションに NOGSTYLE を指定します。GSTYLE システムオプションの詳細については、“GSTYLE System Option” (*SAS/GRAPH: Reference*)を参照してください。

TrueType フォントの移植性

TrueType フォントは異なる動作環境間で移植可能であり、各種の Microsoft Windows 環境でも常に使用可能です。いくつかの TrueType フォントは、UNIX X Windows の一部のバージョンに含まれています。

各国文字のサポート

TrueType フォントは、さまざまな各国文字をサポートしています。TrueType フォントを使用する SAS コードの詳細については、"フォントの指定と各国文字の印刷の例" (279 ページ)を参照してください。

SAS 提供の TrueType フォント

SAS をインストールする際、インストール時の選択によって、複数の TrueType フォントが使用可能になります。SAS で提供される TrueType フォントは、Microsoft と互換性のある Windows Glyph List (WGL) Pan-European 文字セットのフォント、グラフィック文字、アジア多言語、アジア単一言語という 4 つのグループに分類できます。これらのフォントは、Microsoft フォントと同じ形状およびサイズで、フォーマットまたはページングを変更せずに Microsoft フォントの代わりに使用できます。次の表は、SAS フォントと互換 Microsoft フォントを示しています。

表 15.12 Microsoft と互換性のある Windows Glyph List (WGL) Pan-European 文字セットフォント

フォント名	フォントの説明	Microsoft フォントとの互換性
Albany AMT	sans-serif	Arial
Thorndale AMT	serif	Times New Roman
Cumberland AMT	serif fixed*	Courier New

* fixed とは、等幅のことです。

表 15.13 グラフィックシンボルの TrueType フォント

フォント名	フォントの説明	Microsoft フォントとの互換性	Adobe Type1 フォントとの互換性
Symbol MT	192 個の記号	記号	記号
Monotype Sorts*	205 個の Wingdings 文字	なし*	なし*
Arial Symbol	42 個の記号**	なし	なし
Arial Symbol Bold	42 個の記号**	なし	なし
Arial Symbol Bold Italic	42 個の記号**	なし	なし

フォント名	フォントの説明	Microsoft フォントとの互換性	Adobe Type1 フォントとの互換性
Arial Symbol Italic	42 個の記号**	なし	なし
Times New Roman Symbol	42 個の記号**	なし	なし
Times New Roman Symbol Bold	42 個の記号**	なし	なし
Times New Roman Symbol Bold Italic	42 個の記号**	なし	なし
Times New Roman Symbol Italic	42 個の記号**	なし	なし

* SAS Monotype Sorts は、形、記号、装飾グリフから成る装飾用のフォントで、Microsoft TrueType または Adobe Type1 フォントに 1 対 1 でマッピングされてはいません。ただし、SAS Monotype Sorts フォントは、Microsoft "Wingdings" TrueType および Adobe "ITC Zapf Dingbats" Type1 フォントに非常に似ています。

** これらのフォントは、ラテン文字(0、<、=、C、D、L、M、N、P、R、S、U、V、W、X、Z、および a - z)の特殊なグリフを持っています。その他の文字は未定義なので、四角形で表示される場合があります。たとえば、HTML 出力先の場合、Internet Explorer で表示すると、四角形は該当する Latin1 文字に置き換えられます。

表 15.14 *多言語 TrueType フォント*

フォント名	サポートしている言語	フォントの説明
Arial Unicode MS*	アラビア文字、アルメニア文字、基本ラテン文字、ベンガル文字、注音字母、 キリル文字、デーヴァナーガリー文字、グルジア文字、ギリシャ文字とコプト文字、グジャラート文字 グルムキー文字、ハングル字母、ヘブライ文字、平仮名、漢文用記号、カンナダ文字、 片仮名、ラオ文字、マラヤーラム文字、オリヤー文字、タミル文字、テルグ文字、 タイ文字、チベット文字。	sans-serif

フォント名	サポートしている言語	フォントの説明
Times New Roman Uni*	アラビア文字、基本ラテン文字、注音字母、キリル文字、デーヴァナーガリー文字、 グルジア文字、ギリシャ文字とコプト文字、グジャラート文字、ハングル字母、 ヘブライ文字、平仮名、漢文用記号、片仮名、ラオ文字、モンゴル文字、 タミル文字、テルグ文字、タイ文字、チベット文字。	serif

* SAS 9.4 では、Arial Unicode MS フォントおよび Times New Roman フォントは Monotype Sans WT フォントと Thorndale Duospace WT フォントに置き換わりました。

表 15.15 *アジア単一言語の TrueType フォント*

サポートしている言語	フォント名	文字セット
日本語	MS ゴシック、MS UI Gothic、MS P ゴシック	シフト JIS
	MS 明朝、MS P 明朝	シフト JIS
韓国語	Gulim、GulimChe、Dotum、DotumChe	KSC5601
	Batang、BatangChe、Gungsuh、GungsuhChe	KSC5601
簡体字中国語	CSongGB18030C-Light CSongGB18030C-LightHWL MYingHei_18030_C-Medium MYingHei_18030_C-MediumHWL	GB18030 および GB2312
繁体中国語*	HeiT	Big5
	MingLiU、MingLiU_HKSCS、PMingLiU	Big5

* HeiT、MingLiU、MingLiU_HKSCS、および PMingLiu は HKSCS2004 (Hong Kong Supplemental Character Set)文字をサポートしています。

SAS によって提供されたフォント、および Windows にインストール済みのフォントは、SAS をインストールする際に SAS レジストリに自動的に登録されます。UNIX と z/OS にインストール済みのフォントは、SAS をインストールする際に SAS レジストリに手動で登録する必要があります。その他の TrueType フォントの登録については、“フォントの登録” (274 ページ)を参照してください。

フォントの登録

SAS でサポートされているフォント

SAS と一緒にインストールされる TrueType フォントに加え、PostScript Type1 フォントがサポートされています。次の表に、TrueType フォントと Type1 フォントのフォントプレフィックスとファイル拡張子を示します。

表 15.16 サポートされているフォントの種類

種類	タグ	ファイル拡張子
TrueType	<ttf>	.ttf
Type1	<at1>	.pfb *

* .pfm ファイルからのデータに基づき、Windows の SAS/GRAPH SASEMF デバイスと SASWMF デバイスを使用して出力が生成されます。UNIX と z/OS では、.pfm ファイルからのデータに基づき、WMF デバイスと EMF ユニバーサルプリンタを使用して出力が生成されます。このファイルは、PROC FONTREG を使用して Type1 フォントを登録する必要はありません。pfm ファイルを登録していない場合、予想どおりの出力結果にならない可能性があります。

SAS レジストリへのフォントの登録

SAS のインストール時に登録されない TrueType または Type1 フォントを使用するには、FONTREG プロシジャを使用して SAS レジストリにこれらのフォントを登録します。詳細については、“FONTREG” (*Base SAS Procedures Guide*)を参照してください。

注: SAS によって提供されたフォント、および Windows によってインストールされたフォントは、SAS をインストールする際に SAS レジストリに自動的に登録されます。SAS のインストール後にインストールされたフォントは、SAS レジストリに手動で登録する必要があります。

UNIX、Windows、z/OS HFS ファイルシステムでのフォントの登録

次の SAS プログラムをサブミットします。FONTPATH ステートメントはフォントを格納するディレクトリを指定します。ここで、**pathname** はフォントのディレクトリパスです。

```
proc fontreg;
   fontpath 'pathname';
run;
```

SAS レジストリへのフォントの追加の詳細については、“FONTREG” (*Base SAS Procedures Guide*)を参照してください。

注: Microsoft Windows 環境では、TrueType フォントは通常、**C:\WINNT\Fonts** または **C:\Windows\Fonts** ディレクトリにあります。その他のすべての動作環境については、システム管理者に問い合わせて TrueType フォントファイルの場所を確認してください。

詳細については、“FONTREG Procedure” (*Base SAS Procedures Guide*)を参照してください。

z/OS でのフォントの登録

HFS ファイルシステムを使用していない z/OS システムでは、FONTPATH ステートメントのかわりに FONTFILE ステートメントを使用して、フォントを含む固定ブロックシーケンシャルファイルを指定します。MODE=オプションのデフォルト値は ALL であるた

め、次に示す PROC ステートメントは、SAS レジストリに存在していない新しいフォントを追加し、SAS レジストリに存在しているフォントを置き換えます。

```
proc fontreg;
    fontfile 'filename';
run;
```

z/OS 固有

フォントを z/OS システムに追加する場合、固定ブロックレコード形式とレコード長 1 のシーケンシャルデータセットとしてフォントファイルを割り当てる必要があります。

詳細については、“FONTREG Procedure” (*Base SAS Procedures Guide*)を参照してください。

デバイスで登録済みのフォントをリスト表示する

QDEVICE プロシジャを使用すると、FONTREG プロシジャで登録したフォントを含め、SAS レジストリに登録されているフォントのリストを表示できます。特定のデバイスまたはユニバーサルプリンタでサポートされているフォントを表示するには、次のプログラムをサブミットします。

```
/* Macro FONTLIST - Report fonts supported by a device */

%macro fontlist(type, name);
proc qdevice report=font out=fonts;
   &type &name;
   var font ftype fstyle fweight;
run;

data;
   set fonts;
   drop ftype;
   length type $16;
   if ftype = "System"
     then do;
        if substr(font,2,3) = "ttf" then type = "TrueType";
          else if substr(font,2,3) = "at1" then type = "Adobe Type1";
          else if substr(font,2,3) = "cff" then type = "Adobe CFF/Type2";
          else if substr(font,2,3) = "pfr" then type = "Bitstream PFR";
          else type = "System";
        if type ^= "System" then font = substr(font,7,length(font)-6);
          else if substr(font,1,1) = "@" then font = substr(font, 2,length(font)-1);
       end;
       else type = "Printer Resident";
run;

proc sort;
   by font;
run;

title "Fonts Supported by the %upcase(&name) &type";

proc print label;
   label fstyle="Style" fweight="Weigth" font="Font" type="Type";
run;

%mend fontlist;
```

```
%fontlist(printer, pdf)
%fontlist(device, pdf)
%fontlist(device, win)
%fontlist(printer, png)
%fontlist(device, pcl5c)
```

出力データセットの最初の 25 個のフォントの出力を次に示します。

アウトプット 15.3 *PDF プリンタがサポートするフォントのリスト(部分的出力)*

Fonts Supported by the PDF printer

Obs	Font	Style	Weigth	Type
1	Adobe Caslon	Italic	Bold	Printer Resident
2	Adobe Caslon	Italic	Normal	Printer Resident
3	Adobe Caslon	Italic	Semi Bold	Printer Resident
4	Adobe Caslon	Roman	Bold	Printer Resident
5	Adobe Caslon	Roman	Normal	Printer Resident
6	Adobe Caslon	Roman	Semi Bold	Printer Resident
7	Adobe Caslon Oldstyle	Italic	Bold	Printer Resident
8	Adobe Caslon Oldstyle	Italic	Normal	Printer Resident
9	Adobe Caslon Oldstyle	Italic	Semi Bold	Printer Resident
10	Adobe Caslon Oldstyle	Roman	Bold	Printer Resident
11	Adobe Caslon Oldstyle	Roman	Normal	Printer Resident
12	Adobe Caslon Oldstyle	Roman	Semi Bold	Printer Resident
13	Adobe Caslon Small Caps	Roman	Semi Bold	Printer Resident
14	Adobe Caslon Small Caps Oldstyle	Roman	Normal	Printer Resident
15	Adobe Garamond	Italic	Bold	Printer Resident
16	Adobe Garamond	Italic	Normal	Printer Resident
17	Adobe Garamond	Italic	Semi Bold	Printer Resident
18	Adobe Garamond	Roman	Bold	Printer Resident
19	Adobe Garamond	Roman	Normal	Printer Resident
20	Adobe Garamond	Roman	Semi Bold	Printer Resident
21	Adobe Garamond Oldstyle	Italic	Bold	Printer Resident
22	Adobe Garamond Oldstyle	Italic	Normal	Printer Resident
23	Adobe Garamond Oldstyle	Roman	Bold	Printer Resident
24	Adobe Garamond Oldstyle	Roman	Normal	Printer Resident
25	Adobe Garamond Small Caps Oldstyle	Roman	Normal	Printer Resident

詳細については、“QDEVICE” (*Base SAS Procedures Guide*)を参照してください。

フォントの使用

Display Manager を使用したフォントの指定

SAS レジストリを更新すると、新しく登録したフォントが SAS で使用可能になります。ユニバーサル印刷を有効にしたときに対話的にフォントにアクセスするには、次の操作を実行します。

1. **ファイル** ⇨ **印刷**を選択します。
2. PDF または PCL5 などのプリンタを選択します。
3. **プロパティ**ボタンをクリックします。
4. **フォント**タブをクリックします。

   このウィンドウには、フォント、スタイル、ウェイト、サイズ(ポイント)、文字セットのドロップダウンボックスが含まれます。
5. **フォント**ボックスの右側の矢印をクリックし、使用可能なフォントのリストをスクロールします。

   TrueType フォントは、角かっこ(< >)で囲んだ **ttf** という文字で示されます。Type1 フォントは、角かっこ(< >)で囲んだ **at1** という文字で示されます。たとえば、TrueType フォント **Albany AMT** は **<ttf> Albany AMT** としてリストされ、Type1 フォント **Times** は**<at1> Times** としてリストされます。山かっこで囲まれた 3 文字のタグにより、物理プリンタに存在する**<ttf> Symbol** と **Symbol** フォントが区別されます。**<ttf> tag** タグまたは **<at1>**タグがないフォントは、プリンタのメモリに存在します。複数の異なる種類があるフォントを指定する際に、SAS フォントを必ず使用するには、角かっこを付けたフォント構文のみを使用します。たとえば、Symbol フォントの場合は**<ttf> Symbol**.
6. 使用するフォントを選択します。
7. **OK** をクリックして**印刷** ダイアログボックスに戻ります。
8. **OK** をクリックして出力を作成します。

SAS プログラムステートメントを使用したフォントの指定

TITLE ステートメントでフォントを指定できます。たとえば、TrueType フォント Albany AMT を TITLE ステートメントで使用する場合は、SAS プログラムに次のコードを追加します。

```
Title1 f="Albany AMT"  "Text in Albany AMT";
```

TITLE ステートメントで区切り文字としてスラッシュ(/)を使用すると、スタイルやウェイトなどの属性を指定することもできます。

```
Title1 f="Albany AMT/Italic/Bold" "Text in Bold Italic Albany AMT";
```

ODS テンプレートの場合、テキストサイズパラメータの後で属性が指定されます。完全な例については、“PROC PRINT とユーザー定義の ODS テンプレートを使用したフォントの指定” (281 ページ) を参照してください。

注: <ttf>タグは、必要な場合(たとえば、TrueType フォントと別の種類の同名フォントを区別する場合など)にのみ使用します。

SYSPRINTFONT=システムオプションを使用したフォントの指定

SYSPRINTFONT=システムオプションは、プログラムエディタ、ログ、出力ウィンドウなどのウィンドウから印刷する場合のデフォルトフォントを設定します。たとえば、

SYSPRINTFONT=システムオプションを使用すると、次の OPTIONS ステートメントをサブミットすることで Albany AMT フォントで出力を印刷できます。

```
options sysprintfont=("Albany AMT");
```

SYSPRINTFONT=システムオプションを使用して、フォントのウェイトとサイズを指定することもできます。たとえば、次のコードでは、太字斜体で 14 ポイントの Arial フォントを指定します。

```
options sysprintfont=("Arial" bold italic 14);
```

明示的なフォント指定または ODS スタイルでデフォルトフォントをオーバーライドできます。

詳細については、“SYSPRINTFONT= System Option” (*SAS System Options: Reference*)を参照してください。

フォントの斜体化と太字化

一部のフォントファミリは、斜体フォントや太字フォントをサポートしていません。ユーザーがこれらのフォントで斜体や太字を指定すると、SAS System は、フォントの斜体化と太字化機能をサポートしているユニバーサルプリンタ上で、斜体化されたフォントや太字化されたフォントを自動的に生成します。次に示すユニバーサルプリンタは、フォントの斜体化と太字化をサポートしています。

表 15.17 *フォントの斜体化と太字化をサポートするユニバーサルプリンタ*

GIF	PCL
TIFF	PNG
SVG*	

*Internet Explorer および Firefox 上でのフォントの斜体化と太字化はサポートされていません。ただし、Chrome、Opera、Safari の各ブラウザでのフォントの斜体化と太字化はサポートされています。SVG フォントをサポートするブラウザの互換性表を表示するには、http://caniuse.com/#feat=svg-fonts を参照してください。

次に示すユニバーサルプリンタは、フォントの斜体化と太字化をサポートしていません。

- PDF
- EMF
- PostScript

傾き係数の変更

グリフの斜体化をサポートしているフォントにおける傾きの度合いを調整できます。フォントの斜体化は、斜体フォントを持っていないタイプフェイスが対象となります。本物の斜体フォントが、フォントの斜体化の影響を受けることはありません。すべてのユニバーサルプリンタでフォントの傾き係数を変更するには、次のステップに従います。

- コマンドバーに **regedit** と入力するか、または**アプリケーションツールバー**から**ツール** ⇨ **オプション** ⇨ **レジストリエディタ**を選択して、レジストリエディタを開きます。
- **レジストリエディタ**ウィンドウの SAS レジストリパネルで、**CORE/PRINTING/FREETYPE** フォルダを展開します。

- 同フォルダを右クリックして、コンテキストメニューから**倍精度実数値の作成**を選択します。
- **倍精度実数値の編集**ウィンドウの**値の名前**フィールドに **SlantFactor** と入力します。
- 希望の傾き係数値を**値のデータ**フィールドに入力します。デフォルト値は **.25** です。

特定フォントに関して傾き係数を変更するには、SAS レジストリエディタを使用して次のタスクを実行します。

- **レジストリエディタ**ウィンドウの SAS レジストリパネルで、**CORE/PRINTING/FREETYPE/FONTS/<ttf>font-name/Attributes** フォルダを展開します。
- 同フォルダを右クリックして、コンテキストメニューから**倍精度実数値の作成**を選択します。
- **倍精度実数値の編集**ウィンドウの**値の名前**フィールドに **SlantFactor** と入力します。
- 希望の傾き係数値を**値のデータ**フィールドに入力します。デフォルト値は.25 です。

フォントの指定と各国文字の印刷の例

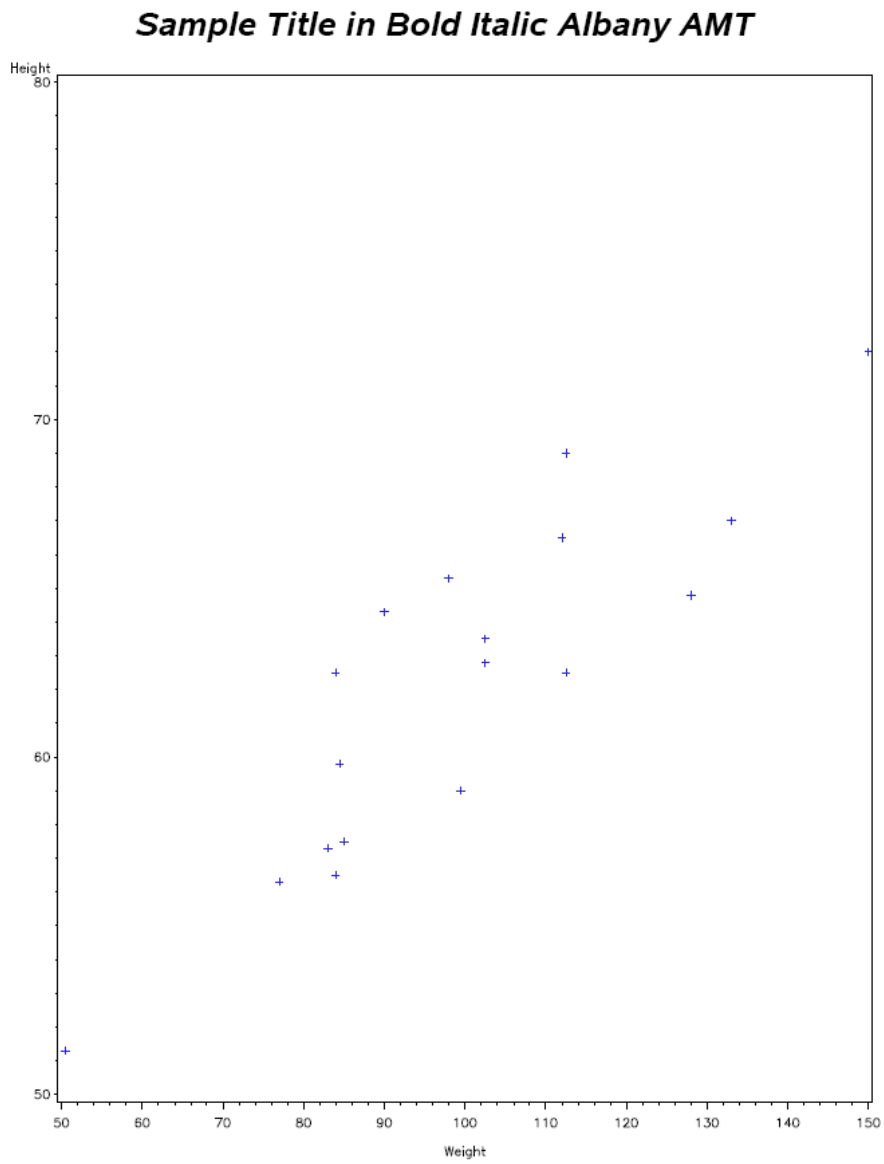
SAS/GRAPH でのフォントの指定

次の例では、太字斜体の Albany AMT フォントのタイトルを持つ出力ファイル **sasprt.pdf** を作成します。

```
options printerpath=pdf device=sasprtc;
ods printer;
title1 color=black f="Albany AMT/Italic/Bold" "Sample Title in Bold Italic Albany AMT";
proc gplot data=sashelp.class;
plot height*weight;
run;
quit;
ods printer close;
```

注: DEVICE=オプションのデフォルト値は、プリンタの種類によって SASPRTM、SASPRTG、SASPRTC のいずれかになります。PRINTERPATH=PCL5 (モノクロプリンタ)の場合、ODS PRINTER のデフォルト値は SASPRTM です。

図 15.25 GPLOT 出力



PROC PRINT でのフォントの指定

次の例では、Albany AMT、Thorndale AMT、および Cumberland AMT フォントのタイトルを持つ出力ファイル **print1.pdf** を生成します。

```
filename new 'print1.pdf';
options printerpath=(PDF new) device=sasprtc obs=5;
ods printer;
proc print data=sashelp.class;
   title1 f='Albany AMT' h=2 'TrueType Albany AMT';
   title2 f='Thorndale AMT' h=3 'Thorndale AMT';
   title3 f='Cumberland AMT' 'Cumberland AMT ';
run;
ods printer close;
```

図 15.26 PROC PRINT を使用した PDF 出力

TrueType Albany AMT
Thorndale AMT
Cumberland AMT

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5

PROC PRINT とユーザー定義の ODS テンプレートを使用したフォントの指定

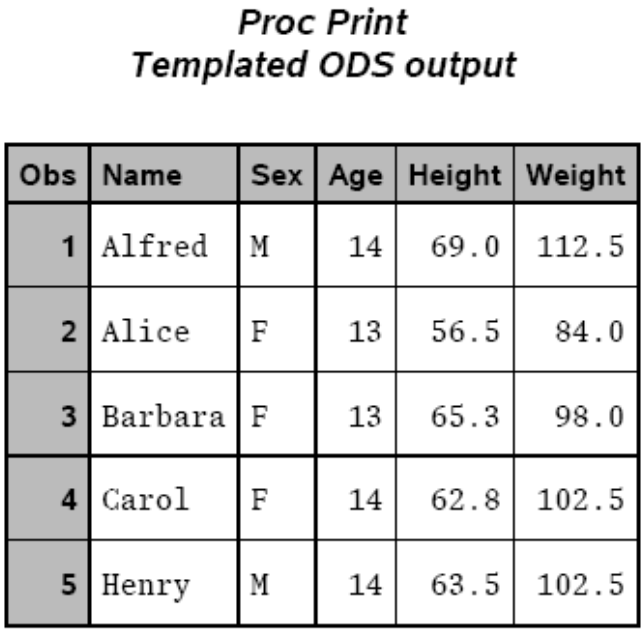
次の例では、フォントスタイルのテンプレートを作成した後、PDF ファイルを生成します。

```
filename out 'print2.pdf';

options printerpath=(pdf out) device=sasprtc;
proc template;
   define style New_style / store = SASUSER.TEMPLAT;
   parent = styles.printer;
   style fonts /
    'docFont'             = ("Cumberland AMT", 12pt)
    'headingFont'         = ("Albany AMT", 10pt, bold)
    'headingEmphasisFont' = ("Albany AMT", 10pt, bold italic)
    'TitleFont'           = ("Albany AMT", 12pt, italic bold)
    'TitleFont2'          = ("Albany AMT", 11pt, italic bold)
    'FixedFont'           = ("Cumberland AMT", 11pt)
    'BatchFixedFont'      = ("Cumberland AMT", 6pt)
    'FixedHeadingFont'    = ("Cumberland AMT", 9pt, bold)
    'FixedStrongFont'     = ("Cumberland AMT", 9pt, bold)
    'FixedEmphasisFont'   = ("Cumberland AMT", 9pt, italic)
    'EmphasisFont'        = ("Albany AMT", 10pt, italic)
    'StrongFont'          = ("Albany AMT", 10pt, bold);
end;
run;

ods printer style=New_style;
proc print data=sashelp.class;
  title1  'Proc Print';
  title2  'Templated ODS output';
run;
ods printer close;
```

図 15.27 PROC PRINT とユーザー定義の ODS テンプレートを使用した PDF 出力



Proc Print
Templated ODS output

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5

各国文字の印刷

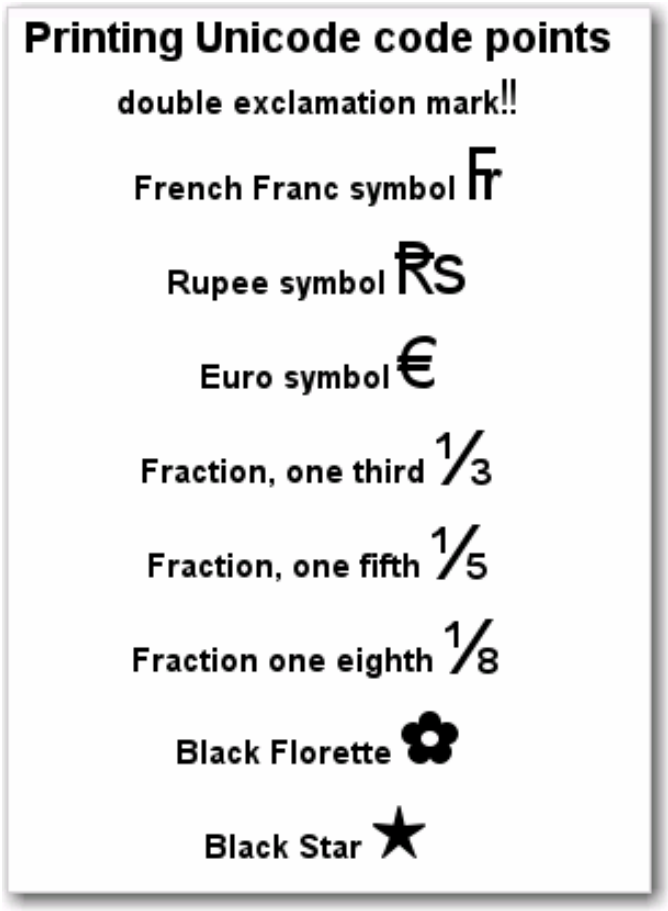
例 1

次の例では、出力ファイル、**titles.png** を生成します。10 個の Unicode 文字が印刷されます。

```
filename new 'titles.png';
options printerpath=(png new)device=sasprtc;
ods printer;

proc gslide;
   title1 "Printing Unicode code points";
   title2 "double exclamation mark" f="Arial Unicode MS/Unicode" h=2 '203C'x;
   title3 "French Franc symbol " f="Arial Unicode MS/Unicode" h=3 '20A3'x;
   title4 "Lira symbol " f="Arial Unicode MS/Unicode" h=3 '20A4'x;
   title4 "Rupee symbol " f="Arial Unicode MS/Unicode" h=3 '20A8'x;
   title5 "Euro symbol " f="Arial Unicode MS/Unicode" h=3 '20Ac'x;
   title6 "Fraction, one third " f="Arial Unicode MS/Unicode" h=3 '2153'x;
   title7 "Fraction, one fifth " f="Arial Unicode MS/Unicode" h=3 '2155'x;
   title8 "Fraction one eighth " f="Arial Unicode MS/Unicode" h=3 '215B'x;
   title9 "Black Florette " f="Arial Unicode MS/Unicode" h=3 '273F'x;
   title10 "Black Star " f="Arial Unicode MS/Unicode" h=3 '2605'x;
run;
quit;
ods printer close;
```

図 15.28 10 個の Unicode 文字の出力



例 2

次の例では、出力ファイル **utf8.gif** を生成します。これは、UTF-8 サーバーで実行する必要があり、使用されている文字を格納する TrueType フォントが必要になります。文字名と関連コードの表は、Unicode Web サイト(**http://www.unicode.org/charts**)にあります。

```
proc template;
   define style utf8_style / store = SASUSER.TEMPLAT;
   parent = styles.printer;
   style fonts /
    'docFont'             = ("Arial Unicode MS", 12pt)
    'headingFont'         = ("Arial Unicode MS", 10pt, bold)
    'headingEmphasisFont' = ("Arial Unicode MS", 10pt, bold italic)
    'TitleFont'           = ("Arial Unicode MS", 12pt, italic bold)
    'TitleFont2'          = ("Arial Unicode MS", 11pt, italic bold)
    'FixedFont'           = ("Times New Roman Uni", 11pt)
    'BatchFixedFont'      = ("Times New Roman Uni", 6pt)
    'FixedHeadingFont'    = ("Times New Roman Uni", 9pt, bold)
    'FixedStrongFont'     = ("Times New Roman Uni", 9pt, bold)
    'FixedEmphasisFont'   = ("Times New Roman Uni", 9pt, italic)
    'EmphasisFont'        = ("Arial Unicode MS", 10pt, italic)
    'StrongFont'          = ("Arial Unicode MS", 10pt, bold);
end;
run;
```

```
%macro utf8chr(ucs2);
 kcvt(&ucs2, 'ucs2b', 'utf8');
%mend utf8chr;
%macro namechar(name, char);
 name="&name"; code=upcase("&char"); char=%utf8chr("&char"x); output;
%mend namechar;

data uft8char;
length name $40;
%namechar(Registered Sign, 00AE);
%namechar(Cent Sign, 00A2);
%namechar(Pound Sign, 00A3);
%namechar(Currency Sign, 00A4);
%namechar(Yen Sign, 00A5);
%namechar(Rupee Sign, 20A8);
%namechar(Euro Sign, 20Ac);
%namechar(Dong Sign, 20Ab);
%namechar(Euro-currency Sign, 20A0);
%namechar(Colon Sign, 20A1);
%namechar(Cruzeiro Sign, 20A2);
%namechar(French Franc Sign, 20A3);
%namechar(Lira Sign, 20A4);
run;

options printerpath=(gif out) device=sasprtc;
filename out 'utf8.gif';

ods printer style=utf8_style;
proc print;
run;
ods printer close;
```

注: このコードの実行時に“テンプレートストアに書き込めません”というエラーが出る場合には、PROC ステップの前に次のコードを入れます。

```
PROC TEMPLATE step:
ODS PATH work.templat(update) sasuser.templat(read)
              sashelp.tmplmst(read);
```

このステートメントにより、テンプレートは、サーバーが読み取りアクセスと書き込みアクセスできる WORK ライブラリに置かれます。

図 15.29 各国文字の出力

Obs	name	code	char
1	Registered Sign	00AE	Â®
2	Cent Sign	00A2	Â¢
3	Pound Sign	00A3	Â£
4	Currency Sign	00A4	Â¤
5	Yen Sign	00A5	Â¥
6	Rupee Sign	20A8	â‚¨
7	Euro Sign	20AC	â‚¬
8	Dong Sign	20AB	â‚«
9	Euro-currency Sign	20A0	â‚
10	Colon Sign	20A1	â‚¡
11	Cruzeiro Sign	20A2	â‚¢
12	French Franc Sign	20A3	â‚£
13	Lira Sign	20A4	â‚¤

ユニバーサル印刷を用いた EMF (Enhanced Metafile Format)グラフィックの作成

SAS における EMF グラフィックの処理

EMF (Enhanced Metafile Format)グラフィックは、カラーグラフィックを生成する、スケーラブルなベクトルグラフィックです。EMF グラフィックをサポートするアプリケーションは Windows で実行されています。デフォルト出力サイズ 800x600 ピクセル、デフォルト解像度 96 dpi では、画面解像度に近い出力になります。EMF グラフィックは、デバイスに依存せず、EMF ビューアで表示されます。

ユニバーサル印刷では、3 つの EMF メタファイル形式、EMF、EMFPlus、EMFDual がサポートされています。次の表では、EMF ユニバーサルプリンタとそれに対応する EMF メタファイル形式を示します。

EMF ユニバーサルプリンタ	EMF メタファイル形式の種類	説明
EMF	EMFPlus	EMF ユニバーサルプリンタは、EMFPlus レコードにあるコマンド、オブジェクトおよびプロパティを使用するグラフィックを作成します。EMFPlus レコードは、EMF レコードとは異なる構造を持ち、EMF とは異なるコマンド、オブジェクトおよびプロパティを使用します。出力の.emf ファイルには、EMF レコードは含まれず、コメントレコード内に埋め込まれた EMFPlus レコードのみが含まれます。 EMFPlus ユニバーサルプリンタは TrueType フォントのみをサポートします。他の種類のフォントを指定した場合、TrueType フォントにフォントがマップされます。
EMFDual	EMFDual	EMFDual ユニバーサルプリンタは、EMF グラフィックと EMFPlus グラフィックの両方を同じ出力ファイルに作成します。表示されるファイルは、EMF ビューアのサポートにより決定されます。 EMFDual プリンタからの出力は、EMF と EMFPlus 形式の両方が含まれるため、大きくなります。
SASEMF	EMF	SASEMF ユニバーサルプリンタは、描画コマンド、オブジェクト定義とプロパティを含む EMF レコードを作成します。アルファチャネルカラーサポートは画像の場合にのみ有効です。ベクターグラフィックスには有効ではありません。SASEMF ユニバーサルプリンタは、SAS の以前のリリースでは、EMF ユニバーサルプリンタと同じです。

EMFDual ユニバーサルプリンタおよび SASEMF ユニバーサルプリンタは、TrueType フォントと Type1 フォントをサポートします。EMF ユニバーサルプリンタは TrueType フォントのみをサポートします。フォントを登録するには、FONTREG プロシジャを使用します。ユニバーサルプリンタが EMF のとき、他の種類のフォントを指定した場合は TrueType フォントにマップされます。

圧縮とフォント埋め込みはサポートされていません。

EMF プリンタの説明を取得するには、次の QDEVICE プロシジャをサブミットして、SAS ログの出力を表示します。

```
proc qdevice;
   printer emf-universal-printer;
run;
```

EMF プリンタは複数ページドキュメントをサポートしていません。プロシジャが複数ページを作成する場合や、複数のプロシジャが ODS PRINTER 出力のコード内で使用される場合には、最初のページのみが表示可能です。

関連項目:

“ユニバーサルプリンタのカラーサポート” (234 ページ)

EMF グラフィックの作成

ODS PRINTER ステートメントを使用すると、スタンドアロン EMF グラフィックを作成できます。EMF ユニバーサルプリンタを、PRINTERPATH=システムオプションの値か、ODS PRINTER ステートメントの PRINTER=オプションの値のいずれかとして指定します。次のサンプルコードでは、EMF ユニバーサルプリンタを ODS PRINTER ステートメントの PRINTER=オプションの値として指定します。

```
ods html close;
ods printer printer=emf;

...more SAS code...

ods printer close;
ods html;
```

現在のディレクトリに、sasprt.emf ファイルが作成されます。

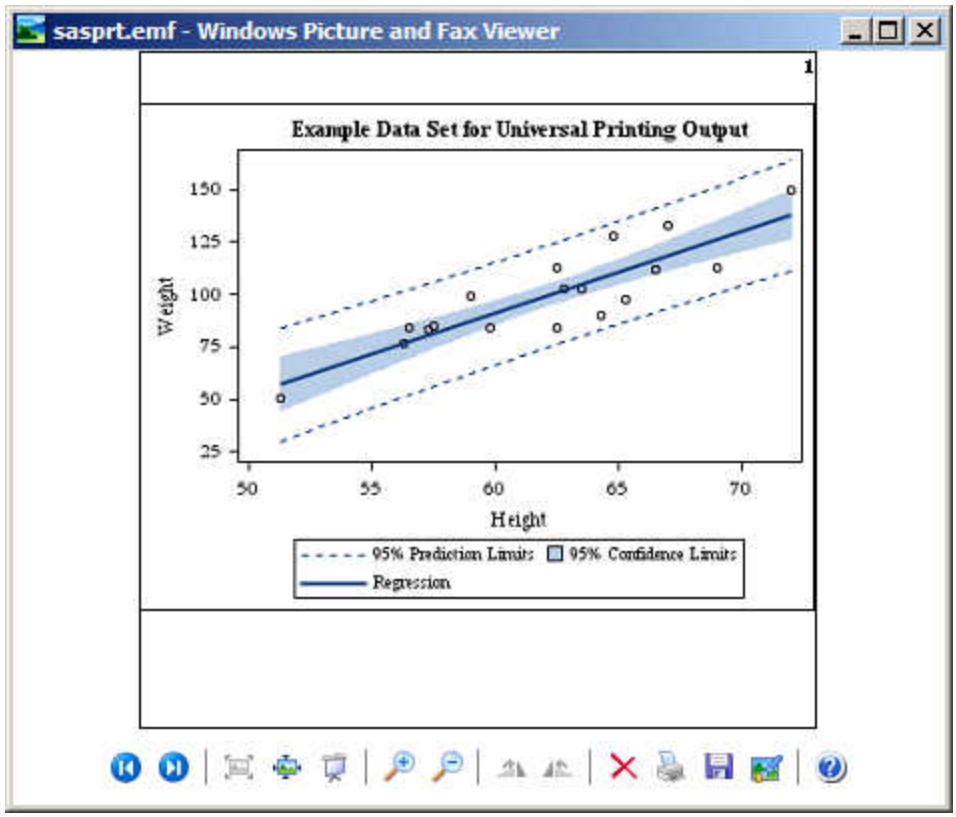
ODS PRINTER ステートメントを用いた EMF グラフィックの作成例

例のデータセット Sashelp.Class および SGPLOT プロシジャを使用すると、次の ODS PRINTER ステートメントは、EMF ファイル sasprt.emf を現在のディレクトリに出力します。

```
options printerpath=emf papersize=("4in" "4in") nodate;
ods html close;
ods printer;
proc sgplot data=sashelp.class;
  reg x=height y=weight / CLM CLI;
run;
ods printer close;
ods html;
```

次の出力は、Windows 画像と FAX ビューアで表示される EMF メタファイルです。

図 15.30 Sashelp.Class の内容を含む EMF (Enhanced Metafile Format)メタファイル



ユニバーサル印刷を用いた GIF 画像の作成

SAS における GIF 画像の処理

GIF (Graphic Interchange Format)は、Web のみで使用される画像形式です。GIF プリンタは、RGBA カラー、アニメーション、透過性をサポートし、FreeType エンジンを使用してフォントを表示します。デフォルト出力サイズは 800x600 ピクセルです。GIF プリンタの説明を取得するには、SAS レジストリでプリンタを表示するか、次の QDEVICE プロシジャをサブミットして、SAS ログに出力を表示します。

```
proc qdevice;
   printer gif;
run;
```

GIF プリンタは複数ページドキュメントをサポートしていません。プロシジャが複数ページを作成する場合や、複数のプロシジャが ODS PRINTER 出力のコード内で使用される場合には、最初のページのみが表示可能です。

関連項目:

- “ユニバーサルプリンタのカラーサポート” (234 ページ)
- “アニメーション GIF 画像と SVG ドキュメントの作成” (329 ページ)

GIF 画像の作成

ODS PRINTER ステートメントを使用して GIF 画像を作成できます。GIF ユニバーサルプリンタを、PRINTERPATH=システムオプションの値か、ODS PRINTER ステートメントの PRINTER=オプションの値のいずれかとして指定します。次のサンプルコードでは、GIF ユニバーサルプリンタを ODS PRINTER ステートメントの PRINTER=オプションの値として指定します。

```
ods html close;
ods printer printer=gif;

...more SAS code...

ods printer close;
ods html;
```

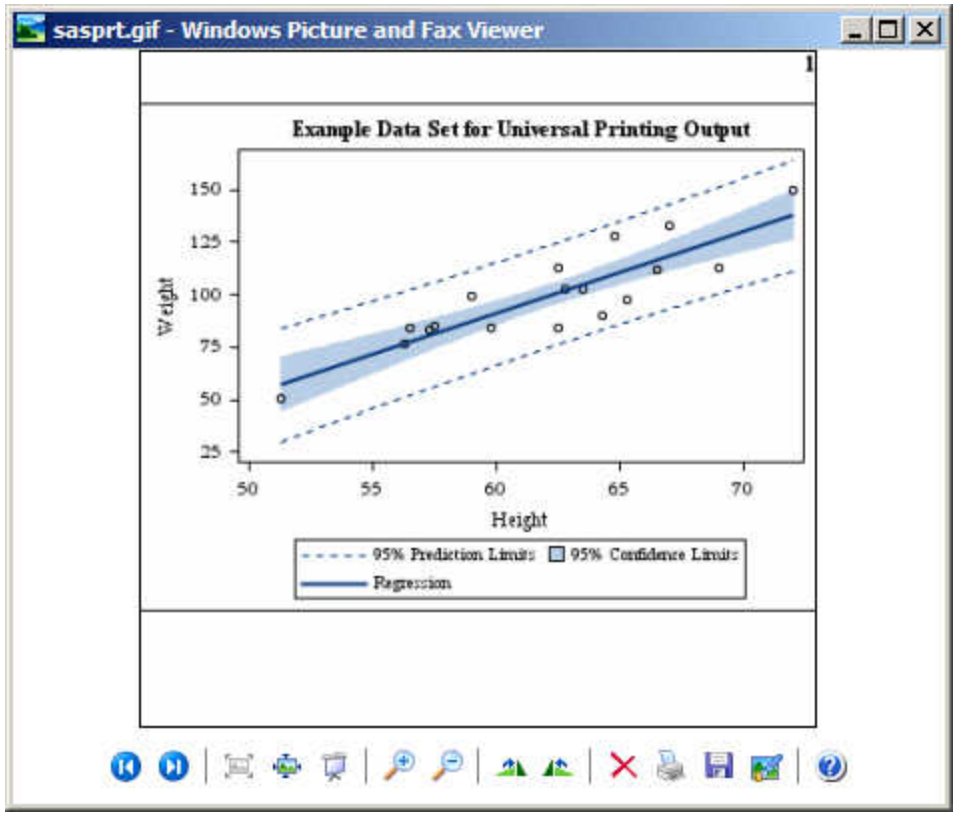
現在のディレクトリに、sasprt.gif ファイルが作成されます。

ODS PRINTER ステートメントを用いた GIF 画像の作成例

例のデータセット Sashelp.Class および SGPLOT プロシジャを使用すると、次の ODS PRINTER ステートメントは、GIF 画像 sasprt.gif を現在のディレクトリに出力します。

```
options printerpath=gif papersize=("4in" "4in") nodate;
ods html close;
ods printer;
proc sgplot data=sashelp.class;
  reg x=height y=weight / CLM CLI;
run;
ods printer close;
ods html;
```

次に、Windows 画像と FAX ビューアに表示した GIF 画像を示します。



ユニバーサル印刷を用いた PCL (Printer Command Language)ファイルの作成

SAS における PCL ファイルの処理

PCL は、Hewlett-Packard (HP)によって開発され、さまざまな印刷デバイスの幅広いプリンタ機能を制御するためにアプリケーションが使用する言語です。ユニバーサル印刷で作成された PCL ファイルは、HP LaserJet プリンタおよび HP Color LaserJet プリンタに送信できます。ユニバーサル印刷 PCL プリンタには、PCL4、PCL5、PCL5c、PCL5e プリンタが含まれます。

- PCL4 は、PCL 4 言語をサポートするレガシー Hewlett-Packard プリンタで印刷されるモノクロ出力を生成します。
- PCL5 は、PCL 5 言語をサポートする Hewlett-Packard プリンタで印刷されるモノクロ出力を生成します。
- PCL5c は、PCL 5c 言語をサポートする Hewlett-Packard プリンタで印刷されるカラー出力を生成します。
- PCL5e は、デフォルトで 600 dpi でモノクロ出力を生成し、PCL 5e 言語をサポートする Hewlett-Packard プリンタで印刷されます。

PCL プリンタの説明を取得するには、SAS レジストリでプリンタを表示するか、次の QDEVICE プロシジャをサブミットして、SAS ログに出力を表示します。

```
proc qdevice;
printer pcl-printer-name;
```

```
run;
```

関連項目:

"ユニバーサルプリンタのカラーサポート" (234 ページ)

PCL ファイルの作成

ODS PCL ステートメントまたは ODS PRINTER ステートメントを使用して PCL ファイルを作成できます。ODS PCL は、デフォルトでは PCL5 ユニバーサルプリンタを使用します。別の PCL プリンタを指定するには、ODS PCL ステートメントで PRINTER=の値を設定します。*pcl-printer* ユニバーサルプリンタを、PRINTERPATH=システムオプションの値か、ODS PRINTER ステートメントの PRINTER=オプションの値のいずれかとして指定します。PRINTERPATH=*pcl-printer* システムオプションを設定した場合は、ODS PRINTER ステートメントで *pcl-printer* を指定する必要はありません。

次に、PCL ファイルを作成するコードの例をいくつか示します。最初のサンプルでは、ODS PCL ステートメントでユニバーサルプリンタを指定せず、デフォルトの PCL5 プリンタを使用します。2 番目のサンプルでは、PCL5 ユニバーサルプリンタを ODS PCL ステートメントの PRINTER=オプションの値として指定します。

- ```
  ods html close;
  ods pcl;

  ...more SAS code...

  ods pcl close;
  ods html;
  ```
- ```
  ods html close;
  ods pcl printer=pcl5;

  ...more SAS code...

  ods pcl close;
  ods html;
  ```

:

同じサンプルコードを使用し、ODS PCL を ODS PRINTER に置き換えると、PCL ファイルを作成できます。

- ```
  ods html close;
  ods printer printer=pcl5c;

  ...more SAS code...

  ods printer close;
  ods html;
  ```
- ```
  options printerpath=pcl5c;
  ods html close;
  ods printer;

  ...more SAS code...

  ods printerl close;
  ods html;
  ```

現在のディレクトリに、sasprt.pcl ファイルが作成されます。PCL ファイルは、Hewlett-Packard LaserJet プリンタまたは Hewlett-Packard Color LaserJet プリンタに出力を送信することで、作成後に表示できます。PCL ファイルは、一部のソフトウェアアプリケーションではモニタに表示することもできます。

ユニバーサル印刷を用いた PDF ファイルの作成

SAS における PDF ファイルの処理

PDF ファイルは、Adobe Acrobat Reader およびその他のアプリケーションで読み込むことができます。SAS では、ODS (Output Delivery System)で PDF ファイルを作成します。ODS は、PDF ユニバーサル印刷プリンタを使用して PDF を作成します。ODS には、ドキュメントに適用できるスタイルとテンプレートがあります。または、独自のスタイルとテンプレートを作成して、ドキュメントをカスタマイズできます。詳細については、“ODS PDF Statement ” (*SAS Output Delivery System: User's Guide*)を参照してください。

PDF プリンタの説明を取得するには、SAS レジストリでプリンタを表示するか、次の QDEVICE プロシジャをサブミットして、SAS ログに出力を表示します。

```
proc qdevice;
   printer pdf;
run;
```

注: SAS/GRAPH をインストールしている場合は、PDF 出力には、リンクやポップアップテキストボックスを含めることができます。詳細については、“Enhancing Web Presentations with Chart Descriptions, Data Tips, and Drill-Down Functionality ” (*SAS/GRAPH: Reference*)を参照してください。

PDF ファイルの作成

ODS PDF ステートメントまたは ODS PRINTER ステートメントを使用して PDF ファイルを作成できます。PDF ユニバーサルプリンタを、PRINTERPATH=システムオプションの値か、ODS PRINTER ステートメントの PRINTER=オプションの値のいずれかとして指定します。ODS PDF ステートメントは、PDF ユニバーサルプリンタを使用して出力を作成します。そのため、ODS PDF ステートメントを使用する際に、PDF ユニバーサルプリンタを明示的に指定する必要はありません。

次に、PDF ファイルを作成するコードの例をいくつか示します。最初のサンプルでは、ODS PDF ステートメントが PDF ユニバーサルプリンタを使用して PDF を作成するので、PDF ユニバーサルプリンタを指定する必要はありません。2 番目のサンプルでは、PDF ユニバーサルプリンタが PRINTERPATH=システムオプションの値として指定され、ODS PRINTER ステートメントで PDF が作成されます。

- ```
  ods html close;
  ods pdf;

  ...more SAS code...

  ods pdf close;
  ods html;
  ```
- ```
  options printerpath=pdf;
  ods html close;
  ods printer;
  ```

```
...more SAS code...

ods printer close;
ods html;
```

現在のディレクトリに sasprt.pdf ファイルが作成され、**結果ビューア**ウィンドウで PDF が開かれます。

ODS PDF ステートメントを使用して PDF を作成する例

この例では、データセット Sashelp.Class の最初から 5 つのオブザベーションを含む PDF ファイルを作成します。

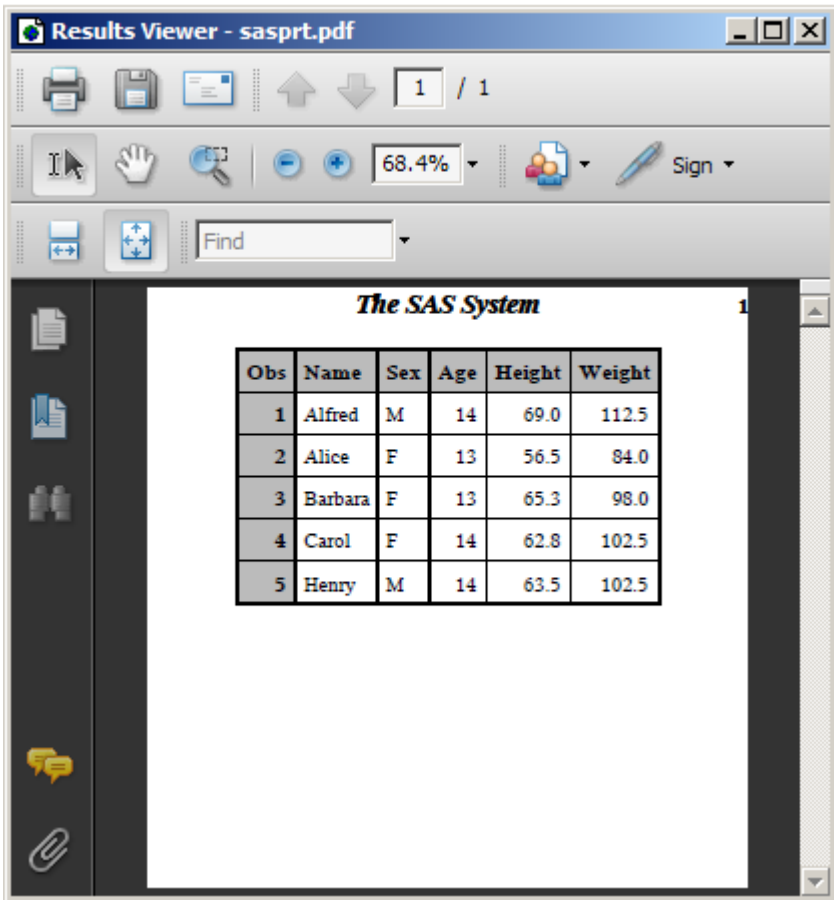
```
options obs=5 nodate pageno=1;
ods html close;
ods pdf;

proc print data=sashelp.class;
run;

ods pdf close;
ods html;
```

次に、PDF 出力結果を示します。

図 15.31 *Sashelp.Class の内容を含む PDF ファイル*



PDF 出力に影響するシステムオプション

PDF 出力を作成する前に、SAS システムオプションを使用してドキュメントのセキュリティ制約を設定できます。ドキュメントのセキュリティ制約では、ドキュメントに対して可能な処理に加えて、セキュリティ方式、印刷の解像度、暗号化レベルを指定します。

次の表は、PDF ドキュメントのセキュリティ制約を設定するために使用できるシステムオプションをリストしています。

システムオプション	説明
PDFACCESS	PDF ドキュメントからテキストやグラフィックが、視覚障害のある読者のためにスクリーンリーダーで読み込めるかどうかを指定します。
PDFASSEMBLY	PDF ドキュメントがアセンブルできるかどうかを指定します。
PDFCOMMENT	PDF ドキュメントのコメントが変更できるかどうかを指定します。
PDFCONTENT	PDF ドキュメントのコンテンツを変更できるかどうかを指定します。
PDFCOPY	PDF ドキュメントからテキストやグラフィックをコピーできるかどうかを指定します。
PDFFILLIN	書き込み可能な PDF 形式かどうかを指定します。
PDFPAGELAYOUT	PDF ドキュメントのページレイアウトを指定します。
PDFPAGEVIEW	PDF ドキュメントのページの表示モードを指定します。
PDFPASSWORD	PDF ドキュメントを開くために使用するパスワード、および PDF ドキュメント所有者が使用するパスワードを指定します。
PDFPRINT	PDF ドキュメントを印刷する解像度を指定します。
PDFSECURITY	PDF ドキュメントの暗号化レベルを指定します。

ユニバーサル印刷を使用した PNG (Portable Network Graphics)ファイルの作成

SAS における PNG (Portable Network Graphics)の処理

PNG (Portable Network Graphics)は、World Wide Web で表示される GIF および TIFF 画像形式に置き換わるものとして設計された画像形式です。SAS ユニバーサルプリンタまたは SAS/GRAPH デバイスドライバで作成された PNG 画像は、PNG 参照ライブ

ラリ(Libpng)を使用します。PNG は、ODS HTML 出力先および SAS/GRAPH のグラフィックス出力のためのデフォルト形式です。

PNG プリンタの説明を取得するには、SAS レジストリでプリンタを表示するか、次の QDEVICE プロシジャをサブミットして、SAS ログに出力を表示します。

```
proc qdevice;
   printer png;
run;
```

関連項目:

“ユニバーサルプリンタのカラーサポート” (234 ページ)

PNG ユニバーサルプリンタ

SAS では、3 つの PNG ユニバーサルプリンタを使用できます。

表 15.18 *SAS で使用できる PNG ユニバーサルプリンタ*

プリンタ名	説明
PNG	PNG 画像を 96 dpi で生成します
PNGt	透明な背景とともに PNG 画像を 96 dpi で生成します
PNG300	PNG 画像を 300 dpi で生成します

PNG プリンタは複数ページドキュメントをサポートしていません。プロシジャが複数ページを作成する場合や、複数のプロシジャが ODS PRINTER 出力のコード内で使用される場合には、最初のページのみが表示可能です。

PNG 画像の作成

ODS PRINTER ステートメントを使用すると、PNG 画像を作成できます。PNG ユニバーサルプリンタを、PRINTERPATH=システムオプションの値か、ODS PRINTER ステートメントの PRINTER=オプションの値として指定します。

次に、PNG 画像を作成するためのサンプルコードを示します。

```
ods html close;
ods printer printer=png;

...more SAS code...

ods printer close;
ods html;
```

現在のディレクトリに、sasprt.png ファイルが作成されます。

SAS/GRAPH では、PNG デバイスは PNG ユニバーサルプリンタへのショートカットです。SAS/GRAPH デバイスによる PNG 画像の作成については、*SAS/GRAPH: Reference* を参照してください。

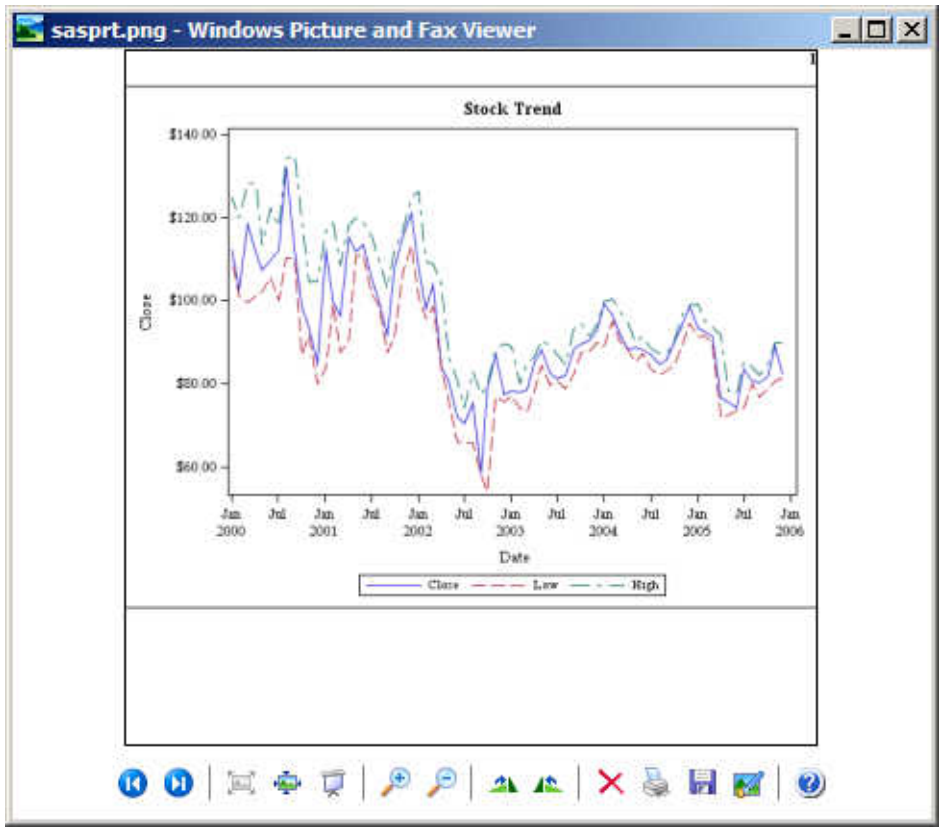
ODS PRINTER ステートメントを用いた PNG ファイルの作成例

PNG ユニバーサルプリンタのいずれかを使用して SAS で PNG 画像を作成するには、PRINTERPATH=システムオプションおよび ODS PRINTER ステートメントで、次の例のように PNG プリンタを指定します。

```
options printerpath=png nodate;
ods html close;
ods printer;
proc sgplot data=sashelp.stocks
  (where=(date >= "01jan2000"d and stock = "IBM"));
  title "Stock Trend";
  series x=date y=close;
  series x=date y=low;
  series x=date y=high;
run;
ods printer close;
ods html;
```

次の出力は、Windows 画像と FAX ビューアで表示される PNG グラフィックです。

図 15.32 ODS プリンタを使用して作成した PNG 画像



PNG ファイルをサポートする Web ブラウザとビューア

次のブラウザおよびビューアは、指定したバージョン以降であれば、ほとんどの PNG 画像機能をサポートしています。

- Microsoft Internet Explorer 7.01b
- Mozilla Firefox 1.5.0.4
- Netscape Navigator 6
- IrfanView for Windows
- Microsoft Photo Editor
- Windows 画像と FAX ビューア

PNG 画像をサポートするブラウザやビューアの詳細については、PNG Web ページ www.libpng.org を参照してください。

ユニバーサル印刷を使用した PostScript ファイルの作成

SAS における PostScript ファイルの処理

ユニバーサル印刷は、PostScript プリンタのいくつかのレベルをサポートしています。デフォルト PostScript プリンタとデフォルトユニバーサルプリンタは、PostScript Level 1 カラープリンタです。ODS (Output Delivery System)を使用して PostScript ファイルを作成します。

PostScript ユニバーサルプリンタの説明を取得するには、SAS レジストリでプリンタを表示するか、次の QDEVICE プロシジャをサブミットして、SAS ログに出力を表示します。

```
proc qdevice;
   printer postscript;
run;
```

PostScript 出力は、透過 GIF ファイルをサポートしています。PostScript ファイルを表示するには、Ghostview を使用できます。Acrobat Distiller がインストールされている場合は、PostScript ファイルを変換して、Adobe Reader で表示可能な PDF ファイルを作成できます。

関連項目:

“ユニバーサルプリンタのカラーサポート” (234 ページ)

PostScript ファイルの作成

ODS PS ステートメントまたは ODS PRINTER ステートメントを使用して PostScript ファイルを作成できます。PS ユニバーサルプリンタを、PRINTERPATH=システムオプションの値か、ODS PRINTER ステートメントの PRINTER=オプションの値のいずれかとして指定します。ODS PS ステートメントは、PS ユニバーサルプリンタを使用して出力を作成します。そのため、ODS PS ステートメントを使用する際に、PS ユニバーサルプリンタを明示的に指定する必要はありません。

次に、PS ファイルを作成するコードの例をいくつか示します。最初のサンプルでは、ODS PS ステートメントが PS ユニバーサルプリンタを使用して PS ファイルを作成するので、PS ユニバーサルプリンタを指定する必要はありません。2 番目のサンプルでは、PS ユニバーサルプリンタが PRINTERPATH=システムオプションの値として指定され、ODS PRINTER ステートメントで PS ファイルが作成されます。

- `ods html close;`

```
ods ps;

...more SAS code...

ods ps close;
ods html;
```

- ```
  options printerpath=ps;
  ods html close;
  ods printer;

  ...more SAS code...

  ods printer close;
  ods html;
  ```

現在のディレクトリに、sasprt.ps ファイルが作成されます。

ODS PS ステートメントを用いた PostScript ファイルの作成例

この例では、データセット Sashelp.Class の最初から 5 つのオブザベーションを含む PS ファイルを作成します。

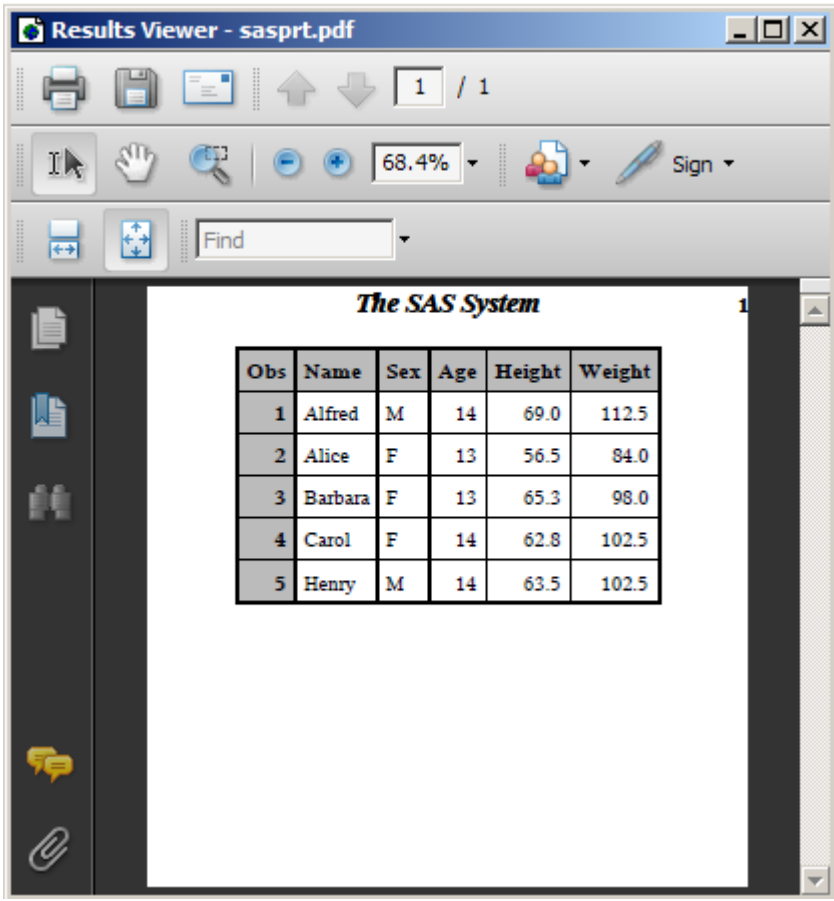
```
options obs=5 nodate pageno=1;
ods html close;
ods ps;

proc print data=sashelp.class;
run;

ods ps close;
ods html;
```

PDF 出力結果に変換した PostScript ファイルを次に示します。
```

図 15.33 Sashelp.Class の内容を含む PDF ファイル



Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5

ユニバーサル印刷を用いた SVG (Scalable Vector Graphics)ファイルの作成

SAS における SVG (Scalable Vector Graphics)の概要

SAS における SVG (Scalable Vector Graphics)の処理

SVG (Scalable Vector Graphics)は、2 次元ベクトルグラフィックスを記述するための XML 言語です。SAS では、SVG ドキュメントの用の W3C 勧告に基づいて SVG ドキュメントが作成されます。SAS SVG ドキュメントは、UNICODE 標準エンコーディング UTF-8 で作成されます。

SAS は、ユニバーサルプリンタと SAS/GRAPH デバイスドライバで SVG ドキュメントを作成できます。SAS/GRAPH SVG デバイスドライバは SVG ユニバーサルプリンタを使用するため、このセクションでは、SAS/GRAPH を使用する SVG ドキュメントの作成についての情報がある程度含まれています。

SAS では、SVG ユニバーサルプリンタとデバイスドライバがグラフの作成によく使用されます。グラフは ODS Graphics または SAS/GRAPH を使用して作成できます。SVG ユニバーサルプリンタを使用して、SVG ドキュメントとして作成した表やレポートを表示することもできます。

いくつかの ODS 出力先(EPUB、HTML、HTML5、LISTING、PRINTER 出力先)は、SVG ドキュメントの作成に使用できます。SVG は、ODS HTML5 出力先のデフォルトユニバーサルプリンタとデバイスドライバです。

SVG ドキュメントは、スタンドアロンファイルにするか、または HTML5 や EPUB ファイル内に統合できます。スタンドアロン SVG ドキュメントは、リンクのターゲットとして参照したり、HTML ドキュメント内に埋め込まれたファイルとして参照したり、CSS2 または XSL プロパティとして参照したりできます。Web ページへの SVG ドキュメントの埋め込みについては、W3 SVG Web サイト(http://www.w3.org/TR/SVG)の SVG 1.1 仕様に記載されている、SVG ドキュメントを Web ページで使用する場合のトピックを参照してください。

複数ページ SVG ドキュメントでは、Base SAS および SAS/GRAPH でアニメーションを入れられます。ODS Graphics プロシジャを指定せずにユニバーサル印刷を使用して Base SAS でアニメーション SVG ドキュメントを作成する場合、アニメーション SVG ドキュメントはスライドショーやアニメーション PowerPoint プレゼンテーションとして表示されます。詳細については、“アニメーション GIF 画像と SVG ドキュメントの作成” (329 ページ)を参照してください。

SAS/GRAPH がインストールされている場合、SVG ドキュメントにはリンクやポップアップテキストボックスを含めることができます。

ここに記載した情報は、Base SAS および ODS Graphics でユニバーサルプリンタを使用して SVG ドキュメントを作成する場合に限られます。SAS/GRAPH で SVG ファイルを作成することの詳細については、“Enhancing Web Presentations with Chart Descriptions, Data Tips, and Drill-Down Functionality ” (*SAS/GRAPH: Reference*)を参照してください。

SVG 規格の詳細については、W3 ドキュメント(http://www.w3.org/TR/SVG)を参照してください。

SVG 用語

SVG キャンバス
: SVG ドキュメントが表示されるスペースです。

viewBox
: ビューポートに表示される SVG ドキュメントの座標系と領域を指定します。

ビューポート
: SVG ドキュメントが表示される SVG キャンバス内の有限矩形スペースです。SAS では、スケーラブルビューポートの場合は PAPERSIZE=システムオプションの値、静的ビューポートの場合は SVGWIDTH=および SVGHEIGHT=システムオプションで決まります。

ビューポート座標系またはビューポート領域
: ビューポートの起点 X 座標と Y 座標、幅と高さの値です。

ユーザー座標系またはユーザー領域
: ビューポートに表示するドキュメントの領域の起点 X 座標と Y 座標、幅と高さの値です。

ユーザー単位
: 環境の座標系で定義された測定単位と同じものです。多くの座標系では、ピクセルを使用します。環境で使用される測定単位を判別するには、システム管理者に問い合わせて確認してください。

SVG ドキュメントの作成理由について

SVG ドキュメントは、SVG をサポートしている任意のビューアまたはブラウザで、任意のサイズで明確に表示されます。SVG ドキュメントは、コンピュータモニタ、PDA、携帯電話、印刷ドキュメントで表示するドキュメントを作成する場合に適しています。SVG ド

キュメントはベクトルグラフィックなので、見やすさを維持したまま、単独の SVG ドキュメントを任意の画面解像度に変えることができます。一方、複数ラスタグラフィック画像では、さまざまな画面解像度とサイズで画像を表示するために、複数の異なる画面解像度を使用する必要がある場合もあります。

SVG ドキュメントは、GIF や PNG など、ラスタグラフィックユニバーサルプリンタで作成された同じ画像よりも、ファイルサイズが小さい場合もあります。

SVG ドキュメントの Web サーバーコンテンツタイプ

Web サーバーの MIME コンテンツタイプ設定に SVG ドキュメント用の適切な設定がない場合、SVG ドキュメントが Web ブラウザではテキストファイルで表示されるか、読み込めない可能性があります。

SVG ドキュメントが正しく表示されるようにするには、次の MIME コンテンツタイプを使用するように Web サーバーを構成します。

```
image/svg+xml
```

SVG ユニバーサルプリンタとその出力

次の表に、SAS がサポートする SVG ユニバーサルプリンタを示します。

表 15.19 *SVG ユニバーサルプリンタ*

プリンタ名	説明
SVG *	SVG 1.1 ドキュメントを作成します。
SVGt *	透明な(背景がない)SVG 1.1 ドキュメントを作成します。
SVGnotip	ツールチップのない SVG 1.1 ドキュメントを作成します。
SVGZ *	圧縮 SVG 1.1 ドキュメントを作成します。
SVGView *	SVG ファイルに複数ページが含まれている場合、ナビゲーションコントロールを含んだ SVG1.1 ドキュメントを作成します。

* このプリンタを SAS/GRAPH で使用する場合、ポップアップデータチップを作成できます。詳細については、“Data Tips for Web Presentations” (*SAS/GRAPH: Reference*)を参照してください。

プリンタを作成するための SVG プロトタイプは、SAS レジストリの CORE\PRINTING\PROTOTYPES にあります。PRTDEF プロシジャで独自の SVG プリンタを定義できます。詳細については、“PRTDEF” (*Base SAS Procedures Guide*)および “PRTDEF プロシジャを用いたユニバーサルプリンタの管理” (261 ページ)を参照してください。

SVG プリンタの説明を取得するには、SAS レジストリでプリンタを表示するか、次の QDEVICE プロシジャをサブミットして、SAS ログに出力を表示します。

```
proc qdevice;
   printer svg-printer-name;
run;
```

関連項目:

- “ユニバーサルプリンタのカラーサポート” (234 ページ)
- “アニメーション GIF 画像と SVG ドキュメントの作成” (329 ページ)

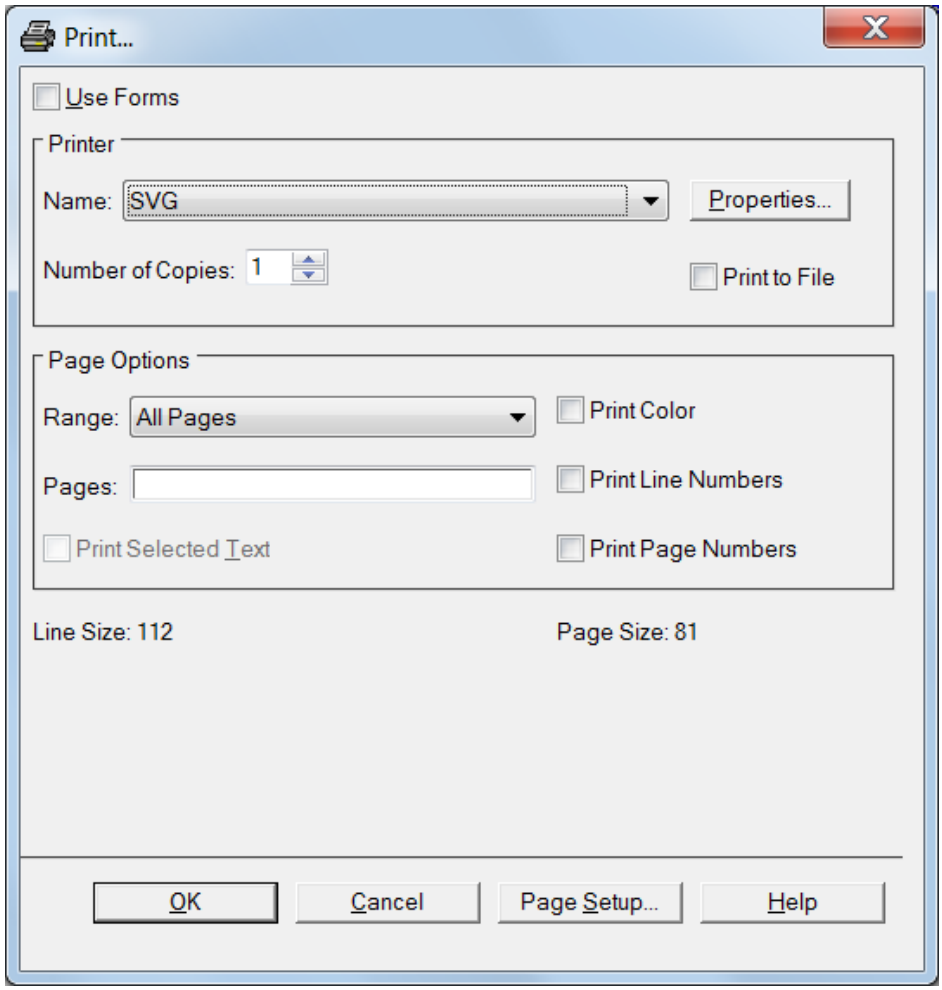
SVG ドキュメントを作成する方法

ユニバーサルプリンタを用いた SVG ドキュメントの作成の基本事項

SVG ドキュメントを作成するには、**印刷**ダイアログボックスまたはプログラミングステートメントを使用します。

印刷ダイアログボックスで SVG ドキュメントを作成するには、名前ドロップダウンリストボックスから SVG プリンタを選択します。

図 15.34 *SVG ドキュメントを印刷するための印刷ダイアログボックス*



プログラムで SVG ドキュメントを作成するには、PRINTERPATH=システムオプションの値として SVG ユニバーサルプリンタを指定します。さらに、次に示すような ODS PRINTER ステートメントなどで ODS 出力先を指定します。

```
options printerpath=svg;
ods html close;
ods printer;
```

```
...more SAS code...

ods printer close;
ods html;
```

または、ODS PRINTER ステートメントで SVG プリンタを指定すれば、次のように OPTIONS ステートメントを省略できます。

```
ods html close;
ods printer printer=svg;

...more SAS code...

ods printer close;
ods html;
```

SAS/GRAPH を使用して SVG グラフを作成するには、ODS LISTING ステートメントを使用できます。

これらのステートメントを使用して、ODS Graphics の SVG グラフを作成できます。

- ```
  ods html5 options (svg_mode="inline");
  ods graphics /imagefmt=svg;

  ...more SAS code...

  ods html5 close;
  ```
- ```
  options printerpath=svg;
  ods html;

  ...more SAS code...

  ods html close;
  ```
- ```
  ods listing;
  ods graphics /imagefmt=svg;

  ...more SAS code...

  ods listing close;
  ```
- SAS/GRAPH を使用する場合:

  ```
  ods listing;
  goptions dev=SVG;

  ...more SAS code...

  ods listing close;
  ```

SAS には、SVG ドキュメントのさまざまな側面を変更できる複数のシステムオプションがあります。SVG ドキュメントの特質をいくつか次に示します。

- 特定の SVG ユニバーサルプリンタ
- SVG ドキュメントの高さと幅
- viewBox のサイズ
- マルチページ SVG ドキュメントにナビゲーションコントロールを含めるかどうか
  ```

ODS PRINTER ステートメントの NEWFILE オプションを使用すると、各プロシジャまたは DATA ステップから出力用の SVG ドキュメントを作成できます。

詳細については、次の言語要素を参照してください。

- “PRINTERPATH= System Option” (*SAS System Options: Reference*)
- “ODS PRINTER Statement ” (*SAS Output Delivery System: User's Guide*)
- SVG ドキュメントのシステムオプション (308 ページ)

SVG ドキュメントを作成する ODS 出力先の要約

次の表に、SVG ドキュメントの作成に使用できる ODS 出力先と作成される出力を示します。

給紙先	SVG ドキュメントの種類	作成される出力
ODS EPUB *	ODS Graphics と SAS/GRAPH によって作成されたグラフ	統合 SVG ドキュメントが付属した EPUB ファイル
ODS HTML	ODS Graphics と SAS/GRAPH によって作成されたグラフ	グラフと HTML ファイルごとの SVG ファイル
ODS HTML5 svg_mode='inline' *	ODS Graphics と SAS/GRAPH によって作成されたグラフ	統合 SVG ドキュメントが付属した HTML ファイル
ODS HTML5 svg_mode='embed' *	ODS Graphics と SAS/GRAPH によって作成されたグラフ	グラフと HTML ファイルごとの SVG ファイル
ODS LISTING	ODS Graphics と SAS/GRAPH によって作成されたグラフ	グラフごとの SVG ファイル
ODS PRINTER	DATA ステップと SAS プロシジャによって作成された出力すべて	ODS PRINTER と ODS PRINTER CLOSE の間に作成された全出力を 1 つにした SVG ファイル

* SVG がデフォルトのプリンタです。

注: ODS Graphics が作成するグラフは、GOPTIONS SAS/GRAPH ステートメントで指定されたオプションを使用しません。GOPTIONS ステートメントは SAS/GRAPH にのみ有効です。

ODS Graphics プロシジャを使用して作成された SVG ファイルのデフォルトのファイル名には、接頭辞にプロシジャ名が付加されます。たとえば、PROC SGPLOT 出力のデフォルトのファイル名は sgplot01.svg です。ODS PRINTER を使用して作成された SVG ファイルのデフォルトのファイル名は、sasprt.svg です。

SVG ドキュメントを表示するブラウザサポート

SVG ドキュメントをサポートするブラウザ

SVG ドキュメントを表示するには、SVG (Scalable Vector Graphics)をサポートしているビューアまたはブラウザが必要です。Mozilla Firefox などのブラウザには、SVG ドキュメントのビルトインサポートがあります。

次の表に、SVG ドキュメントをサポートしているブラウザとビューアをいくつか示します。

表 15.20 SVG ブラウザサポート

ブラウザまたはビューア	会社
Batik SVG Toolkit	Apache Software Foundation
eSVG Viewer および IDE	eSVG Viewer for PC、PDA、Mobile
Google Chrome *	Google Inc.
GPAC Project	GPAC
Internet Explorer 9 またはそれ以降*	Microsoft
Mozilla Firefox *	Mozilla Foundation
Opera	Opera Software ASA
Safari (iPad を含む)*	Apple, Inc.
TinyLine	TinyLine

* このブラウザは SAS でサポートされています。

Mozilla Firefox の使用に関する注意

- SVGZ ユニバーサルプリンタを使用して圧縮された SVG ドキュメントはサポートされていません。
- ズームおよびパン機能は現在、実装されていません。
- **表示** ⇨ **ページスタイル** ⇨ **スタイルなし**を選択すると、すべてのグラフがブランクの四角形で表示されます。
- Firefox はフォント埋め込みをサポートしていません。SVG ドキュメントのフォントマッピングの問題を防ぐために、NOFONTEMBEDDING システムオプションを設定できます。SVG ドキュメントを作成して Firefox で表示したときに、FONTEMBEDDING オプションが設定されている場合、**Firefox のオプション**ダイアログボックスの**コンテンツ**タブで定義されているデフォルトフォント設定が使用されます。

HTML5 出力における SVG ドキュメントに関する注意

HTML5 出力では、多くのブラウザがコンテナの 100%まで SVG ファイルを拡大縮小しないことから、SVG ドキュメントの height 属性と width 属性を SVG のサイズに設定

します。Google Chrome と Safari ブラウザは、SVG ファイルをコンテナの 100%まで拡大縮小します。

コンテナのサイズまで拡大縮小できる SVG ファイルの拡大縮小を可能にするには、SVGHEIGHT=システムオプションおよび SVGWIDTH=システムオプションを 100%に設定します。

```
options svgheight="100%" svgwidth="100%";
```

ODS HTML5 出力用に作成した SVGZ ドキュメントは、Google Chrome または Opera Web ブラウザを使用してのみ表示可能です。

SVG ドキュメント内の画像

SAS プログラムが、画像を含んでいる SVG ドキュメントを作成する場合、ドキュメントの画像ごとに次の操作が実行されます。

- 指定した画像を PNG 形式に変換します。
- base64 エンコーディングを使用して PNG 画像をエンコードします。
- base64 でエンコードされた PNG 画像を SVG ドキュメントに埋め込みます。

SVG ドキュメントでは、**<image>**要素には、次のように始まる **xlink** 属性があります。

```
xlink:href="
```

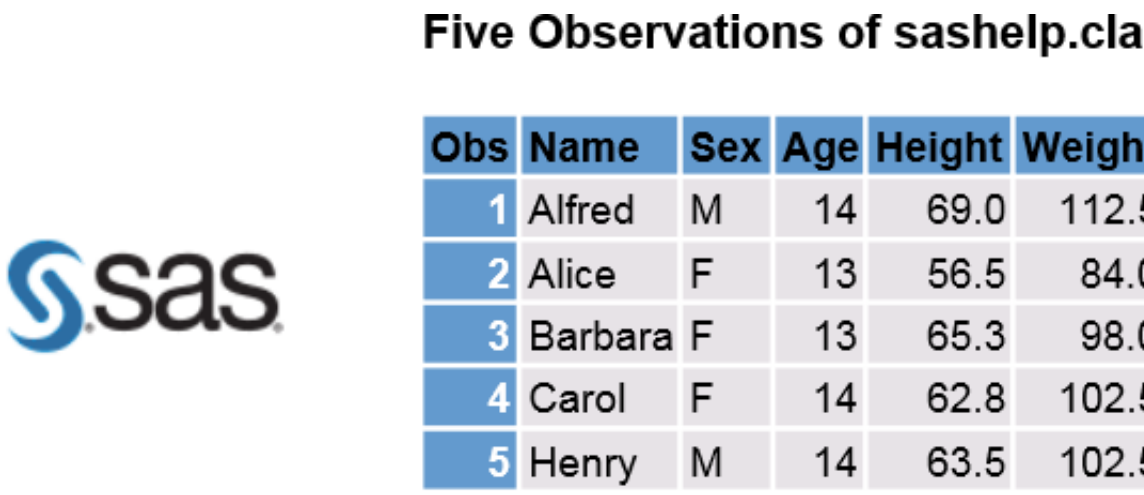
base64 でエンコードされた画像は **base64,** の後に来ます。

デフォルトでは、SVG ユニバーサルプリンタは PNG ファイルをエンコードし、SVG ドキュメントに埋め込みます。次の例では、<image>要素が一番下にあります。エンコードされた画像は、/png;base64,要素属性の後に始まります。画像の最初の文字は **iVBORwOK** です。エンコードされた文字が同じ行で右に伸びていき、このドキュメントでは表示できません。

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" xml:space="pre
<desc></desc>
<svg id="SVGMain_SVGOrig1" viewBox="-1 -1 801 621">
</svg>
<svg id="SVGMain_SVG1" viewBox="-1 -1 801 621">
<defs>
<clipPath id="SVGMain_clipPage1">
<rect x="-1" y="-21" width="801" height="621"/>
</clipPath>
</defs>
<g id="SVGMain_Page1" transform="translate(0,20)" clip-path="url(#SVGMain_clipPage1)">
<rect x="0" y="0" width="800" height="600" style="fill: #E0E0E0; stroke: #E0E0E0; stroke-width: 1
<defs>
<image id="SVGMain_Image1" width="154" height="36" xlink:href="
```

画像はこの SVG ドキュメントの SAS ロゴです。

アウトプット 15.4 *Web ブラウザで表示された SVG ドキュメント*



Five Observations of sashelp.cla

Obs	Name	Sex	Age	Height	Weigh
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5

次のプログラムでは、埋め込み画像が作成されています。

```
proc template;
   define style logo;
      parent = Styles.default;
      style Body from Body /
         backgroundimage = 'c:\saslogo.png';
   end;
run;

ods html close;
options nodate nonumber orientation=landscape obs=5;

ods printer style=logo printer=svgview file='logo.svg';

proc print data=sashelp.class;
   title 'Five Observations of sashelp.class';
run;
ods printer close;
ods html;
```

エンコードされた PNG ファイルを SVG ドキュメントに埋め込むかわりに、SVG ユニバーサルプリンタでは、PNG ファイルを別に作成し、SVG ドキュメント内からその PNG ファイルへのリンクを設定できます。SVG ドキュメントの<image>要素の例を次に示します。

```
<image id="Image3" width="200" height="150" xlink:href="I3svgimg.png">
</image>
```

使用している SVG プリンタで **Images Embedded** レジストリ設定が 0 に設定されている場合、SVG ユニバーサルプリンタは別の PNG ファイルを作成します。

このレジストリ設定を設定するには、次の操作を行います。

1. **レジストリ**エディタを開くには、**ソリューション** ⇨ **アクセサリ** ⇨ **レジストリエディタ**を選択します。
2. レジストリエディタで、**CORE** ⇨ **PRINTING** ⇨ **PRINTERS** ⇨ ***svg-printer*** ⇨ **ADVANCED** を展開します。
3. **Images Embedded** を右クリックし、**Modify** を選択し、**Value Data** を 0 に変更します。

4. **OK** をクリックします。

PNG ファイル名の形式は、*counterPrinterDestinationFilename*.png になります。*counter* は、新しい画像が作成されるたびに 1 ずつ増える整数です。*PrinterDestinationFilename* は、プリンタの出力先ファイル名です。たとえば、デフォルトプリンタ出力先のファイル名 sasprt を使用すると、最初の 3 つの画像は I1sasprt.png、I2sasprt.png、I3sasprt.png となります。

画像へのパスとともに注記が SAS ログに書き込まれます。

スタンドアロンの SVG ドキュメントを作成するための環境設定

スタンドアロンの SVG ドキュメントを作成するための環境設定の概要

“ユニバーサルプリンタを用いた SVG ドキュメントの作成の基本事項” (302 ページ)に示すように、SVG ユニバーサルプリンタは、PRINTERPATH=システムオプションで指定したプリンタ値か、ODS Printer ステートメントのいずれかで指定する必要があります。OPTIONS ステートメントを使用するか、**SAS システムオプション**ウィンドウを使用して SAS が SAS プログラムで呼び出されると、任意の SVG システムオプションを設定できます。

SVG 環境を確立する SVG システムオプション(PRINTERPATH=システムオプションを除く)のデフォルト値を使用すると、SAS SVG ドキュメントを簡単に作成できます。このセクションでは、SVG システムオプションおよびそれがスタンドアロン SVG ドキュメントにどのように影響するかについて説明します。

スタンドアロンの SVG ドキュメントに影響する SAS システムオプション

次のシステムオプションを使用すると、SVG ドキュメントの作成環境を設定できます。

表 15.21 *SVG ドキュメントに影響する SAS システムオプション*

タスク	システムオプション
スタンドアロン SVG ドキュメントを作成するための SVG プリンタの名前を指定します。	PRINTERPATH=
SVG ドキュメントにコメントの埋め込み	COLOPHON
SVG ドキュメントのサイズを設定するオプション	
ユニバーサル印刷に使用する用紙のサイズを設定します。	PAPERSIZE=
SVG ドキュメントの高さを設定します。SVG ドキュメントに埋め込み SVG ドキュメントがある場合、高さの値は最も外側の SVG ドキュメントにのみ影響します。	SVGHEIGHT=
SVG ドキュメントの幅を設定します。SVG ドキュメントに埋め込み SVG ドキュメントがある場合、幅の値は最も外側の SVG ドキュメントにのみ影響します。	SVGWIDTH=
埋め込み SVG ドキュメントの左下隅の X 軸の座標を設定します。	SVGX=

タスク	システムオプション
埋め込み SVG ドキュメントの左下隅の Y 軸の座標を設定します。	SVGY=
X 座標と Y 座標、最も外側の SVG ドキュメント用の viewBox の設定に使用する幅と高さを指定し、ビューポートに表示されるドキュメントの領域の座標を指定します。	SVGVIEWBOX=
SVG ドキュメントの単一の拡大縮小を強制的に適用するかどうかを指定します。	SVGPRESERVEASPECTRATIO=
SVG ドキュメントの外観を変更するオプション	
SVG ドキュメントのタイトルバーに表示されるタイトルを設定します。	SVGTITLE=
マルチページ SVG ドキュメントにナビゲーションコントロールを表示するかどうかを指定します。	SVGCONTROLBUTTONS
SVG ドキュメントで拡大ツールを表示するかどうかを指定します。	SVGMAGNIFYBUTTON
アニメーション SVG ドキュメントのオプション	
アニメーションファイルの作成を開始/停止します。	ANIMATION
アニメーションドキュメントの各フレームが表示される時間を設定します。	ANIMDURATION
アニメーションを通しで連続的にループするか、一度の再生にするかを指定します。	ANIMLOOP
アニメーション内でフレームを重ね合わせるか、順番に再生するかを指定します。	ANIMOVERLAY
Web ブラウザで SVG ドキュメントを表示する場合、アニメーションをすぐに開始するかどうかを指定します。	SVGAUTOPLAY
フレームがフェードインするまでの秒数を設定します。	SVGFADEIN
フレームがフェードイン/アウトを指定されている場合、アニメーションのフレームが前のフレームとオーバーラップするか、または各フレームが順番に再生されるかを指定します。	SVGFADEMODE
フレームがフェードアウトするまでの秒数を設定します。	SVGFADEOUT

詳細については、*SAS System Options: Reference* を参照してください。

SVG ユニバーサルプリンタの設定

PRINTERPATH=システムオプションをいずれかの SVG ユニバーサルプリンタに設定することで、SVG ユニバーサルプリンタを設定します。PRINTERPATH=システムオプションはいつでも設定できます。次の OPTIONS ステートメントで、圧縮 SVG ドキュメントを作成するためのユニバーサルプリンタを設定します。

```
options printerpath=svgz;
```

詳細については、次のトピックを参照してください。

- “SVG ユニバーサルプリンタとその出力” (301 ページ)
- “PRINTERPATH= System Option” (*SAS System Options: Reference*)
- “ユニバーサルプリンタを用いた SVG ドキュメントの作成の基本事項” (302 ページ)

SVG ドキュメントをビューポートに合わせて拡大縮小する

SVG ドキュメントをビューポートに合わせて拡大縮小するには、SVGHEIGHT=システムオプションと SVGWIDTH=システムオプションのデフォルト値 Null を使用します。Null 値は 100%を指定した場合と同じです。つまり、SVG ドキュメントはビューポートのサイズに拡大縮小されます。また、SVGVIEWBOX=システムオプションの値は Null にする必要があります。

詳細については、次のシステムオプション(*SAS System Options: Reference*)を参照してください。

- SVGHEIGHT=システムオプション
- SVGWIDTH=システムオプション
- SVGVIEWBOX=システムオプション

viewBox の設定

<svg>要素の viewBox 属性は、起点 X 座標、起点 Y 座標、SVG ドキュメントの高さ、SVG ドキュメントの幅で構成されます。viewBox 属性値は、SVGVIEWBOX=システムオプションの値を基に設定されます。SVGVIEWBOX=オプションに値がない場合は、PAPERSIZE=システムオプションの値に基づいて、viewBox 属性の高さと幅の引数が設定されます。開始座標の値は 0 に設定されます。

SVGVIEWBOX=、SVGHEIGHT=、SVGWIDTH=システムオプションの値はいずれも Null (デフォルト値)の場合、SVG ドキュメントはビューポートのサイズに合わせて拡大縮小されます。SVGVIEWBOX=システムオプションの値を指定する場合、SVG ドキュメントは、SVGVIEWBOX=オプションで指定した寸法に拡大縮小されます。

SVGHEIGHT=オプションと SVGWIDTH=オプションをパーセントで指定した場合、SVG ドキュメントは、ブラウザウィンドウのサイズが変更された場合にブラウザウィンドウのサイズに合わせて拡大縮小されます。パーセント以外の単位(in、cm、px など)でオプションを指定した場合、SVG ドキュメントのサイズは固定され、ウィンドウのサイズが変わってもブラウザウィンドウに合わせて拡大縮小されません。

詳細については、次のトピックを参照してください。

- “ODS PRINTER Statement ” (*SAS Output Delivery System: User's Guide*)
- “SVGVIEWBOX= System Option” (*SAS System Options: Reference*)
- “PAPERSIZE= System Option” (*SAS System Options: Reference*)
- “静的 viewBox の作成” (311 ページ)

SAS の SVG システムオプションと SVG 要素属性の相互作用

SVGHEIGHT=、SVGWIDTH=、SVGVIEWBOX=、SVGPRESERVEASPECTRATIO=、SVGX=、SVGY=システムオプションの値が、最も外側の<svg>要素の各属性(height、width、viewBox、preserveAspectRatio)の値として使用されます。.たとえば、SVGWIDTH="400"および SVGHEIGHT="300"と指定した場合、<svg>要素が width="400"および height="300"で作成されます。SVGX=システムオプションと SVGY=システムオプションの値は、x 属性や y 属性の<svg>要素にのみ埋め込まれます。

すべてのシステムオプションには Null デフォルト値があります。SVGVIEWBOX=システムオプションが Null の場合、viewBox のサイズは PAPERSIZE=システムオプションの値に基づいて決まります。そのため、システムオプションの値を指定しない場合、SAS に設定される唯一の<svg>属性は、SAS SVG システムオプションを使用している viewBox 属性です。その他の<svg>属性(バージョンや xmlns など)は、SAS によって設定されますが、システムオプションを使用して設定されるものではありません。

すべての SAS SVG システムオプションがデフォルト値に設定された場合、次の<svg>要素が作成されます。

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xml:space="preserve" baseProfile="full" version="1.1"
     id="SVGMain" onload='SVGMain_Init("SVGMain")'
     viewBox="-1 -1 801 601">
```

SVGPRESERVEASPECTRATIO=システムオプションは、<svg>要素の preserveAspectRatio 属性を設定するために使用され、viewBox 属性が SVG ドキュメントでも設定済みの場合にのみ有効です。

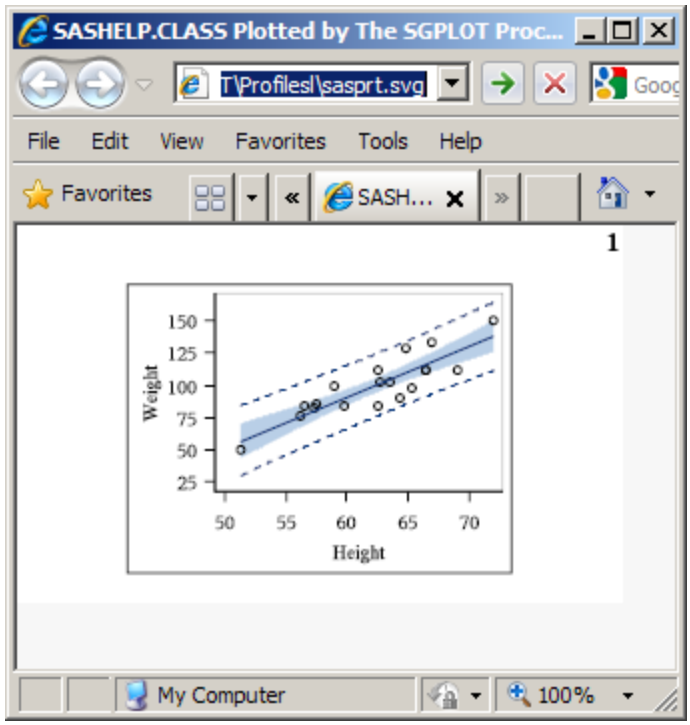
SVG オプションには、負の値を指定できます。ただし、SVGHEIGHT=オプションまたは SVGWIDTH=オプション、または SVGVIEWBOX=オプションの height 引数と width 引数で負の値を指定すると、SVG ドキュメントはブラウザで表示されません。SVG ドキュメントの起点を配置するために、SVGVIEWBOX=オプションの x 引数および y 引数で負の値を指定すると便利な場合があります。SVGVIEWBOX=オプションで負の引数を指定すると、出力結果が右にシフトします。SVGVIEWBOX=オプションで負の値を指定すると、ドキュメントの位置が下にシフトします。

静的 viewBox の作成

静的な viewBox は、変更不可の viewBox です。ブラウザウィンドウのサイズ変更などでビューポートが変更されても、viewBox は同じサイズのままです。静的な viewBox を作成するには、PAPERSIZE=、SVGWIDTH=、SVGHEIGHT=システムオプションと同じ幅と高さの値を指定します。PAPERSIZE=システムオプションによって、viewBox が設定されます。SVGWIDTH=システムオプションと SVGHEIGHT=システムオプションによって、SVG ドキュメントのサイズが設定されます。SVGHEIGHT=および SVGWIDTH=オプションがパーセントで指定された場合、SVG ドキュメントは、ブラウザウィンドウのサイズが変わると、そのブラウザウィンドウに合わせて拡大縮小します。図 15.35 (312 ページ)に、次のシステムオプションで作成される静的な viewBox を表示します。

```
options nodate printerpath=svg papersize=("8cm" "5cm") svgwidth="8cm" svgheight="5cm"
   svgtitle="Sashelp.Class Plotted by The SGPLOT Procedure";
```

図 15.35 静的 viewBox



SVGWIDTH=、SVGHEIGHT=、および SVGPRESERVEASPECTRATIO=システムオプションを Null にリセットするには、一重引用符または二重引用符を 2 つ重ね、間にスペースを入れずに指定します。

```
options printerpath=svg svgwidth="" svgheight="" svgpreserveaspectratio="";
```

縦横比の維持

viewBox のサイズを変更すると、SVGPRESERVEASPECTRATIO=システムオプションを使用して、SVG ドキュメントの縦横比を維持するかどうかを指定し、SVG ドキュメントをビューポートに配置する方法を指定できます。このオプションは、次の割り当てのいずれかを使用して設定します。

SVGPRESERVEASPECTRATIO=*align* | *meetOrSlice* | NONE | “”

SVGPRESERVEASPECTRATIO=“*align meetOrSlice*”

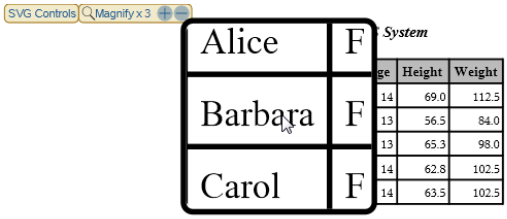
最初の引数 *align* は、使用する配置方法を指定して単一の拡大縮小を強制的に適用するかどうかを指定します。たとえば、**`xMidYMid`** 値を使用すると、viewBox の中間点 X 値ビューポートの中間点 X 値に合わせ、ドキュメントを水平方向の中央に配置します。

2 番目の引数 *meetOrSlice* は、SVG ドキュメントを viewBox に拡大縮小する方法を指定します。この引数の値には、**`meet`** または **`slice`** を指定できます。**`meet`** を指定した場合、SVG ドキュメントは、他の条件に合わせながら可能な限り拡大されます。ビューポートには、未使用のスペースが表示されます。**`slice`** を指定した場合、SVG ドキュメントは、他の条件に合わせながら可能な限り縮小されます。後者の場合、SVG ドキュメントの一部がカットされたように表示されます。SVG ドキュメントは完全な状態で存在していますが、ビューポートではすべて表示されません。ブラウザコントロールを使用すると、ビューポート内で SVG ドキュメントを移動して表示できます。

詳細については、“SVGPRESERVEASPECTRATIO= System Option” (*SAS System Options: Reference*)を参照してください。

拡大ツールを SVG ドキュメントに取り込む

SVGMAGNIFYBUTTON システムオプションを設定することで、拡大ツールを SVG ドキュメントに含むことができます。ツールが有効な場合は拡大鏡が使用可能で SVG ドキュメントの一部を拡大できます。拡大範囲のサイズは変更できません。



デフォルトでは、拡大ツールは SVG ドキュメントに含まれていません。SVGMAGNIFYBUTTON システムオプションを明示的に設定する必要があります。この OPTIONS ステートメントが使用できます。

```
options svgmagnifybutton;
```

拡大ツールを無効にするには、NOSVGMAGNIFYBUTTON システムオプションを使用します。

拡大ツールを有効化すると、**拡大**ボタン 

が、SVG ドキュメントの最上部に表示されます。拡大鏡を表示するには、**拡大**をクリックします。マウスを使用して、SVG ドキュメントの表示域上に拡大鏡を移動し、拡大鏡の下にある領域を拡大します。拡大鏡を止めるには、**拡大**をもう一度クリックします。デフォルトでは、拡大鏡は 3 段階の拡大レベルで領域を拡大します。+をクリックすると拡大レベルが増え、–をクリックすると拡大レベルが減ります。SVG ドキュメントを iPad で表示している場合は、**拡大**ボタンを最初にタップするとツールチップが表示されます。2 回目以降のタップでは、拡大の変更と拡大鏡の終了が行われます。

拡大ツールの使用にはいくつか制約があります。

- 拡大ツールは、SVGT プリンタとアニメーション SVG ドキュメントではサポートされていません。
- SVGnotip プリンタを使用する場合、ツールチップは表示されないため、拡大鏡が有効か無効かを区別できません。
- 拡大ツールが有効な場合、拡大ツールは複数ページドキュメントの**索引**ページで終了できます。

SVG ドキュメントでズームレベルの制御が見込めるブラウザに SVG ドキュメントを表示する場合、拡大ツールは大いに有用です。

詳細については、“SVGMAGNIFYBUTTON System Option” (*SAS System Options: Reference*)を参照してください。

SVG ドキュメントにタイトルを追加する

SVGTITLE=システムオプションを使用すると、ブラウザに SVG ドキュメントのみが表示されている場合に、ウィンドウのタイトルバーにタイトルを追加できます。SVG ドキュメントが HTML ページに埋め込まれている場合、<svg>タグの svgtitle 属性は無効です。前のトピックの静的なビューポートの例では、ブラウザのタイトルバーにタイトルが表示されています。

詳細については、“SVGTITLE= System Option” (*SAS System Options: Reference*)を参照してください。

ODS PRINTER の出力先を用いたスタンドアロンの SVG ドキュメントの作成

SVG ドキュメントの作成

SVG ドキュメントを作成するには、少なくとも PRINTERPATH=システムオプションを SVG ユニバーサルプリンタに設定し、SAS プログラムで ODS PRINTER ステートメントを指定します。または、ODS PRINTER ステートメントで PRINTER=オプションを指定します。

```
options printerpath=svg;
ods html close;
ods printer;
proc sgplot data=sashelp.class;
  reg x=height y=weight / CLM CLI;
run;
ods printer close;
ods html;
```

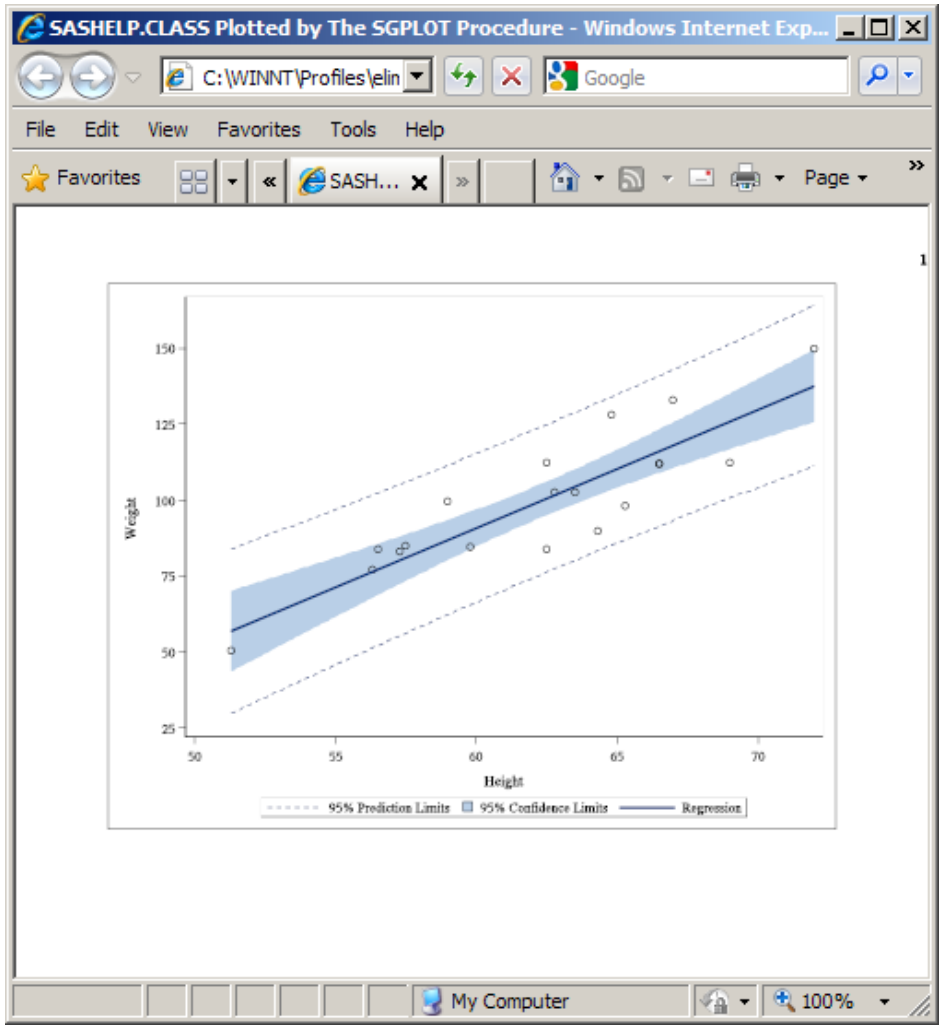
この例では、特定の SVG システムオプション値が SVG ドキュメントのサイズを変更するように設定されていません。そのため、viewBox は、PAPERSIZE=システムオプションで指定したデフォルトサイズです。SVGWIDTH=および SVGHEIGHT=システムオプションで値が指定されていないので、SVG ドキュメントはビューポートに合わせて拡大縮小します。次に示すのは、SAS が作成する<svg>要素です。

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xml:space="preserve" baseProfile="full" version="1.1"
     id="SVGMain" onload='SVGMain_Init("SVGMain")'
     viewBox="-1 -1 801 601">
```

sasprt.svg という単一の SVG ドキュメントが作成され、動作環境に従って特定の場所に格納されます。Windows では、ファイルは現在のディレクトリに格納されます。UNIX では、ファイルはホームディレクトリに格納されます。z/OS では、ファイルは z/OS UNIX System Services の HFS (Hierarchal File System)ファイルまたは z/OS データセットとして格納されます。SVG ファイルが z/OS データセットに書き込まれた場合は、PDSE ライブラリ *userid*.SASPRT.SVG に書き込まれます。ODS PRINTER ステートメントで FILE=オプションを使用すると、別のファイル名を指定できます。

次の図は、Microsoft Internet Explorer 用 Adobe Acrobat SVG プラグインを使用する SVG ファイルです。このファイルは、Sashelp.Class データセットをプロットするために SGPLOT プロシジャを使用して作成されたものです。

図 15.36 SGPLOT プロシジャによって SVG ファイルとしてプロットされた Sashelp.Class



SVG、SVGnotip、SVGt、SVGView、SVGZ ユニバーサルプリンタを使用すると、単一 SVG ドキュメントが作成されます。SVG ドキュメントのサイズによっては、ブラウザに完全な SVG ドキュメントが表示される可能性もあります。ブラウザが SVG ドキュメント表示用のコントロールを備えているかどうかを判別するには、使用しているブラウザのドキュメントを参照してください。Internet Explorer 用 Adobe SVG Viewer プラグインでは、Alt キーと左マウスボタンを押すと、連続する複数ページ SVG ドキュメントの異なるページにパンおよび移動できます。

複数ページの SVG ドキュメントを 1 つのファイルに統合

DATA ステップまたはプロシジャで出力結果の新しいページが作成されると、SVG ドキュメントで新しい SVG ページが作成されます。複数ページを持つファイルまたは、SVG ドキュメントページごとに 1 つのファイルを持つ複数の SVG ファイルが作成されます。SVGCONTROLBUTTONS システムオプションおよび ODS PRINTER ステートメントの NEWFILE=オプションでは、複数ページ SVG ドキュメントが、(ファイルのページをナビゲートするコントロールを備えた)連続する 1 つのファイルか、複数の SVG ファイルかを制御します。

ODS PRINTER ステートメントの NEWFILE=オプションが PAGE 以外の値で、システムオプションの次のセットのいずれかが指定されている場合、SAS は、ナビゲーションコントロールを備えた、単一ファイル、複数ページ SVG ドキュメントを作成します。

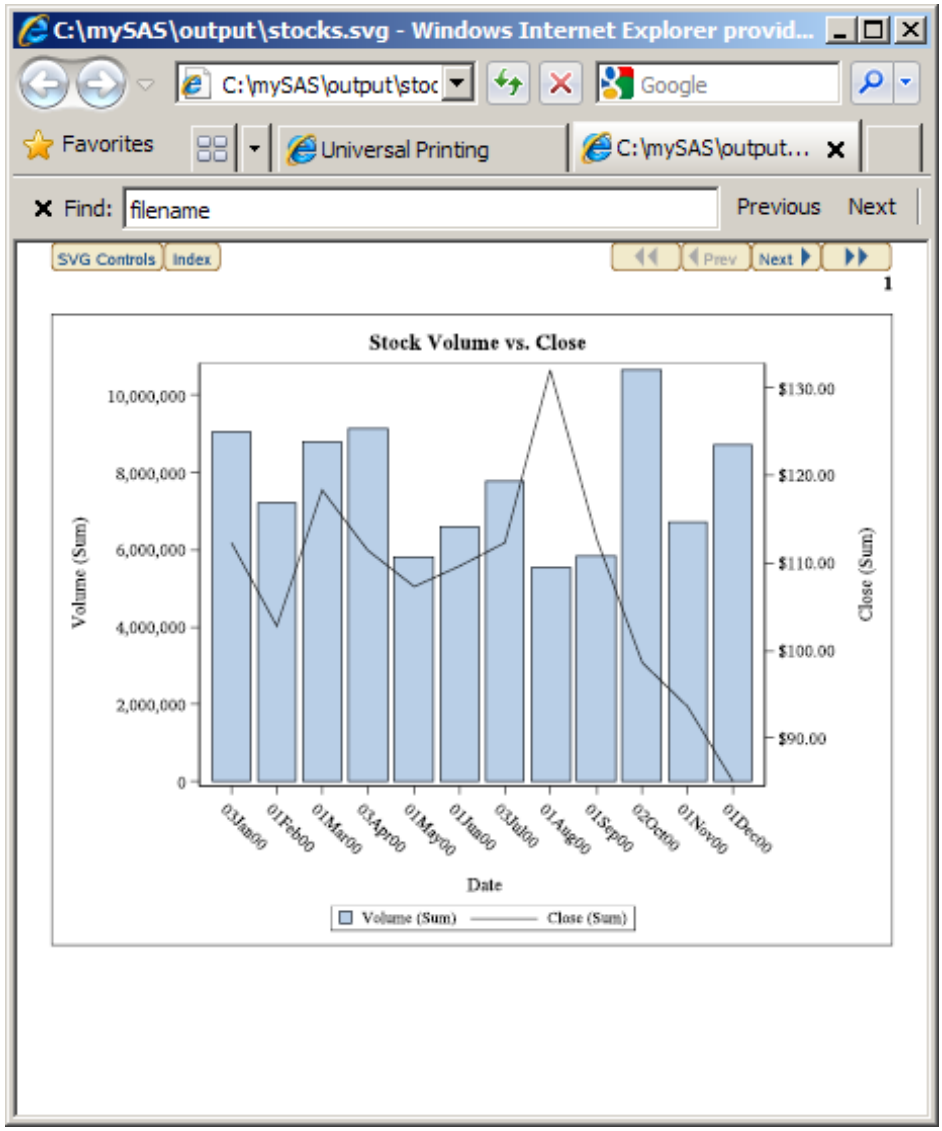
- PRINTERPATH=システムオプションは SVG または SVGZ に設定され、SVGCONTROLBUTTONS システムオプションが設定されます。
- PRINTERPATH=システムオプションが SVGView に設定されます。

SVGView ユニバーサルプリンタでは、SVGCONTROLBUTTONS システムオプションが有効です。

SVGCONTROLBUTTONS システムオプションが指定されていない場合、またはユニバーサルプリンタが SVGView ではない場合、SVG ドキュメントは連続ページレイアウトで作成されます。ドキュメントをナビゲートする場合は、ブラウザコントロールを使用します。

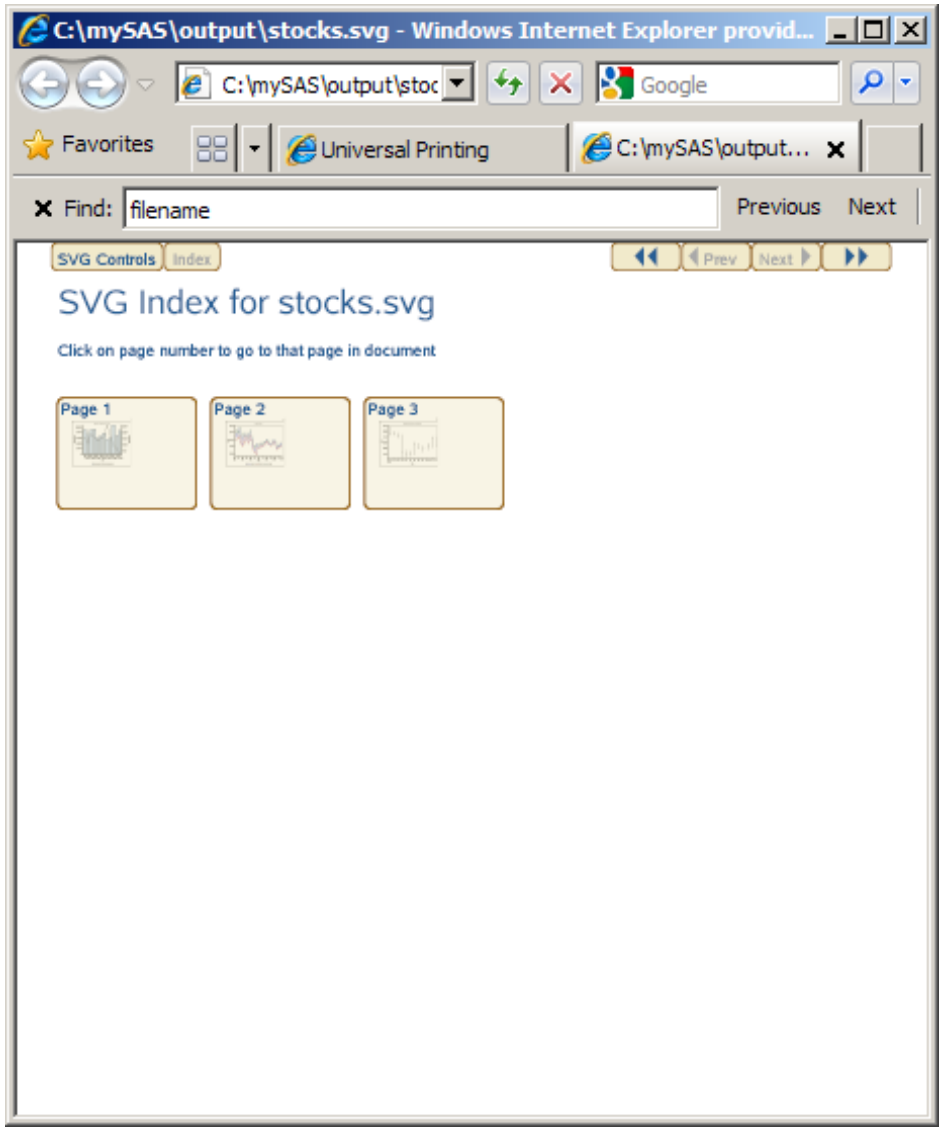
ナビゲーションコントロールを使用すると、次のページ、前のページ、先頭ページ、最終ページに移動したり、全ページのインデックスを表示したり、コントロールの表示と非表示を切り替えたりできます。

図 15.37 *ナビゲーションコントロール付き複数ページ SVG ファイルの先頭ページ*



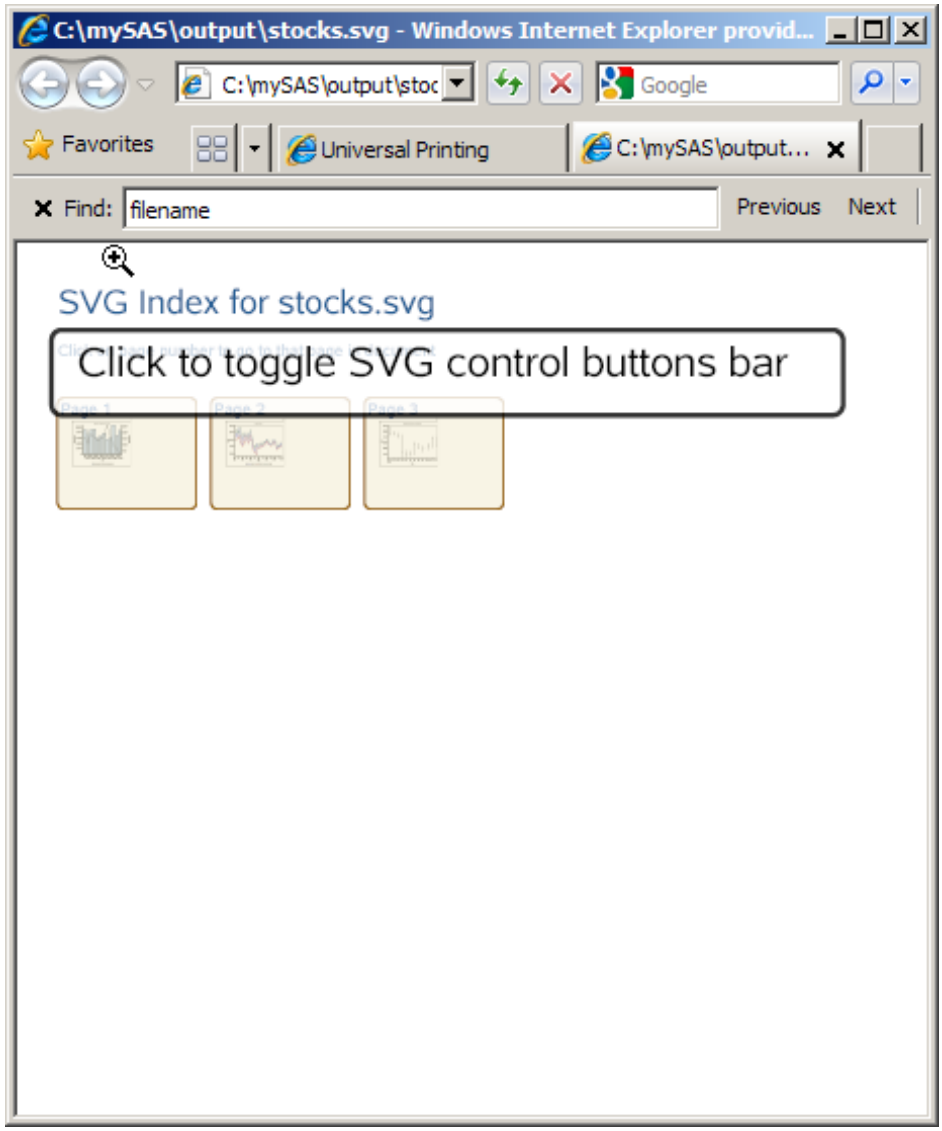
SVG ファイルの全ページのインデックスを表示するには、**インデックス**ボタンを選択します。インデックスから特定のページに移動するには、ページのサムネイル画像を選択します。

図 15.38 ナビゲーションコントロール付き複数ページ SVG ファイルのインデックス



コントロールボタンを非表示にするには、**SVG コントロール**ボタンを選択します。ツールチップは、カーソルがコントロール上に置かれたときに表示されます。ナビゲーションコントロールを再び表示するには、**クリックすると SVG コントロールボタンバーが切り替わります**というツールチップが表示されたときに出力結果の上部領域をクリックします。これは、SVG コントロールの付いていないドキュメントのページを印刷する場合に便利です。

図 15.39 ナビゲーションコントロールを非表示にした複数ページ SVG ファイル



stocks.svg file ファイルを作成した SAS コードを次に示します。

```
options nodate printerpath=(svgview stocks) papersize=("6" "6") ;
filename stocks 'c:\mySas\output\stocks.svg';
ods html close;
ods printer;
proc sgplot data=sashelp.stocks (where=(date >= "01jan2000"d
                                 and date <= "01jan2001"d
                                 and stock = "IBM"));
   title "Stock Volume vs. Close";
   vbar date / response=volume;
   vline date / response=close y2axis;
run;
title;
proc sgplot data=sashelp.stocks
  (where=(date >= "01jan2000"d and stock = "IBM"));
  title "Stock Trend";
  series x=date y=close;
```

```
  series x=date y=low;
  series x=date y=high;
run;
title;
title "Stock High, Low, and Close";
proc sgplot data=sashelp.stocks;
  where Date >= '01JAN2005'd and stock='IBM';
  highlow x=date high=high low=low
    / close=close;
run;
title;
ods printer close;
ods html;
```

NEWFILE=オプションの詳細については、“ODS PRINTER Statement ” (*SAS Output Delivery System: User's Guide*)を参照してください。

複数ページ SVG ファイルのアニメーション作成

SAS システムオプションを使用して、複数ページ SVG ファイルにアニメーションを作成できます。詳細については、“アニメーション GIF 画像と SVG ドキュメントの作成” (329 ページ)を参照してください。

複数ページの SVG ドキュメント用に別々のファイルを作成する

SVG ドキュメントのページごとに個別のファイルを作成するには、ODS PRINTER ステートメントで NEWFILE=PAGE オプションを指定します。プロシジャが明示的に新しいページを開始する際に新しいページが作成され、ページサイズを超えるとページは作成されなくなります。最初のファイルは *filename*.svg という名前になります。それ以降のファイル名には、1 から始まる番号が、*filename*1.svg、*filename*2.svg などのように付加されます。

デフォルトファイル名 sasprt.svg を使用すると、次のコードで 3 つのファイルが作成されます。

- sasprt.svg には、最初の SGPLOT プロシジャからの出力結果が含まれます。
- sasprt1.svg には、2 番目の SGPLOT プロシジャからの出力結果が含まれます。
- sasprt2.svg には、3 番目の SGPLOT プロシジャからの出力結果が含まれます。

```
options nodate printerpath=svgview papersize=("6" "6");
ods html close;
ods printer newfile=page;
proc sgplot data=sashelp.stocks (where=(date >= "01jan2000"d
                                 and date <= "01jan2001"d
                                 and stock = "IBM"));
   title "Stock Volume vs. Close";
   vbar date / response=volume;
   vline date / response=close y2axis;
run;
title;
proc sgplot data=sashelp.stocks
  (where=(date >= "01jan2000"d and stock = "IBM"));
  title "Stock Trend";
  series x=date y=close;
  series x=date y=low;
  series x=date y=high;
run;
title;
```

```
title "Stock High, Low, and Close";
proc sgplot data=sashelp.stocks;
  where Date >= '01JAN2005'd and stock='IBM';
  highlow x=date high=high low=low
    / close=close;
run;
title;
ods printer close;
ods html;
```

NEWFILE=オプションの詳細については、“ODS PRINTER Statement ” (*SAS Output Delivery System: User's Guide*)を参照してください。

透過 SVG ドキュメントを作成して重ね合わせる

SVTt ユニバーサルプリンタを使用すると、透過的で重ね合わせ可能なページがある、透過 SVG ドキュメントを作成できます。次の SAS プログラムは、アメリカ合衆国の地図上に棒グラフを重ね合わせます。

```
data boxanno;
   length function color style $20 text $16;
   retain xsys ysys '2' hsys '3' when 'a';
   set maps.uscity(keep=x y city state);
      where city='Raleigh' and state=stfips('NC');
   color='blue'; size=4; text='V'; position='5'; style='marker'; output;
   myx=x;
   myy=y;
   function='move';
    x=myx; y=myy; output;
   function='draw';
    x=myx-.432; y=myy+.0417; color='black'; line=1; size=.2; style='solid'; output;
   function='move';
    x=myx; y=myy; output;
   function='draw';
    x=myx-.432; y=myy+.178; output;
   function='move';
    x=myx; y=myy; output;
   function='draw';
    x=myx-.251; y=myy+.178; output;
   function='move';
    x=myx; y=myy; output;
   function='draw';
    x=myx-.251; y=myy+.0417; output;

run;

%let name=annomap;
filename odsout '.';

goptions reset=all;
/* Close the HTML and LISTING destinations for map creation. */
ods html close;
ods listing close;
options  printerpath=svgt nodate nonumber;
ods printer file='annomap.svg' ;
```

```
goptions border;

goptions gunit=pct htitle=3 htext=2 ftext="arial/bo"
         iback='c:\public\mySASPrograms\ripple.jpg';
pattern1 v=s c=cornsilk;

title1 c=red "SAS/Graph gmap and Overlayed gchart with printerpath=svgt";
proc gmap data=maps.us map=maps.us ;
id state;
choro state / levels=1 nolegend coutline=blue anno=boxanno
 des="" name="&name";
run;

quit;


goptions iback= hsize=2.07 vsize=1.57 horigin=2.1 vorigin=3.12 autosize=on dev=svgt;

/* you must use the default ods style, for transparency to work */

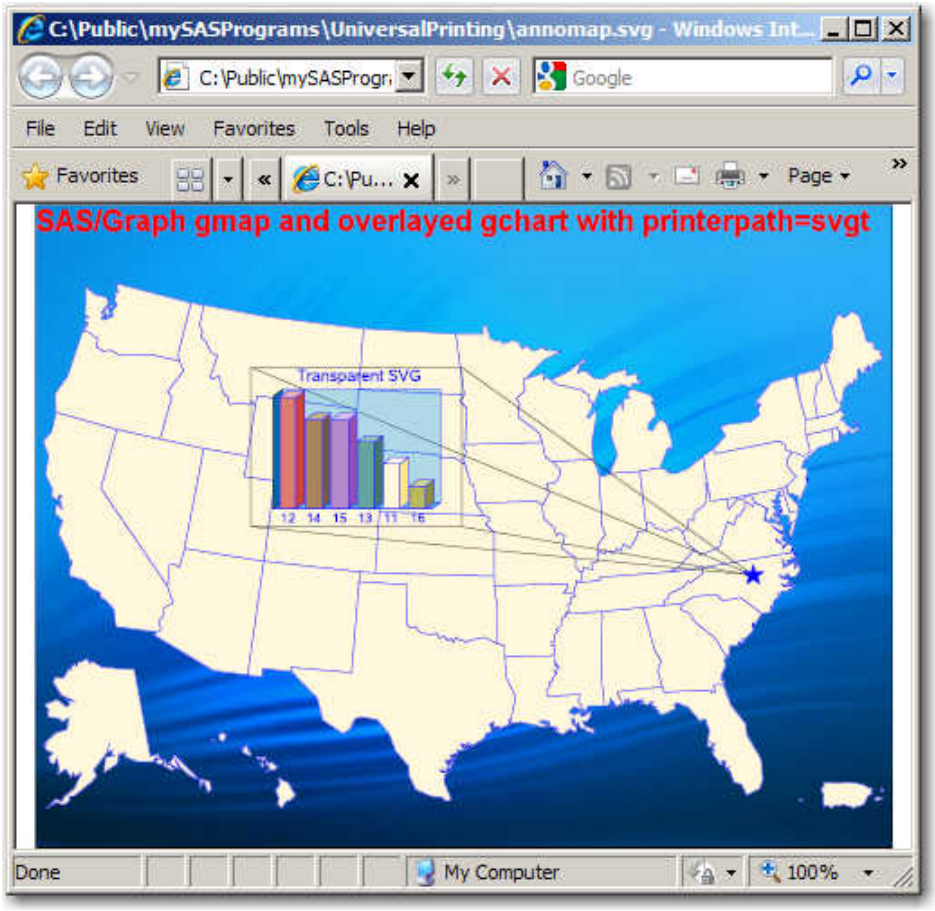
goptions gunit=pct  htext=8 ftext="Albany AMT" ;
title c=blue h=10 'Transparent SVG';
axis1 label=none value=none major=none minor=none style=0;
axis2 color=blue label=none offset=(7,7) value=(color=blue);
proc gchart data=sashelp.class;
   vbar3d age / discrete  patternid=midpoint
   descending raxis=axis1 maxis=axis2 width=9 space=5
   frame cframe=rgba0195FF50 coutline=blue woutline=1
   des="" name="&name.b";
run;

quit;

ods printer close;
```

このコードは次の SVG を作成します。

図 15.40 SAS/GRAPH マップを重ね合わせる棒グラフ



グラフのドリルダウンリンクで重ね合わせた画像を使用する例については、"Enhancing Drill-Down Behavior in SVG Presentations Using HTML Attributes" (*SAS/GRAPH: Reference*)を参照してください。

HTML ファイル内の SVG ドキュメント

HTML ファイル内の SVG ドキュメントの概要

HTML ファイルで SVG ドキュメントを表示するには、SVG ドキュメントへのリンクを作成して HTML ファイル内にその SVG ドキュメントを埋め込むか、または HTML ファイル内に統合される SVG グラフを作成します。

これらの方法を使用して、HTML ファイルに SVG ドキュメントを埋め込むことができます。

- ODS GRAPHICS ステートメントと ODS HTML5 ステートメントでオプション SVG_MODE="EMBED"を使用して、SVG グラフを作成します。
- SAS/GRAPH を使用し、SAS プログラムを ODS HTML DEV=SVG ステートメントを使用して実行します。SVG ドキュメントと HTML ファイルが作成され、<EMBED>要素を使用して SVG ドキュメントが HTML ファイルに埋め込まれます。
- ODS PRINTER ステートメントと PRINTERPATH=SVG オプションを使用して SVG ドキュメントを作成します。次に、<EMBED>要素を使用して、HTML ファイルに SVG ドキュメントを埋め込みます。

ODS HTML5 SVG_MODE='INLINE'ステートメントを使用することで、HTML ファイルに SVG グラフを統合できます。

SAS/GRAPH における SVG ドキュメント作成の詳細については、“Generating SVG, PNG, GIF, and TIFF Graphics ” (*SAS/GRAPH: Reference*)を参照してください。

SVG ドキュメントへのリンク

SVG ドキュメントを HTML ドキュメントにリンクし、SVG システムオプションのデフォルト値を使用する場合、SVG ドキュメントはブラウザウィンドウで開かれ、ウィンドウの表示可能領域のサイズに合わせて拡大縮小します。SVG ドキュメントへリンクする HTML ファイルの例については、図 15.41 (324 ページ)と図 15.42 (325 ページ)を参照してください。

HTML ファイルへの SVG ドキュメントの埋め込み

SVG ドキュメントを HTML ファイルに埋め込むと、<EMBED>タグの height 属性と width 属性がビューポートの寸法になります。SVG ドキュメントを作成する際に SVG システムオプションのデフォルト値を使用する場合、SVG ドキュメントはビューポートのサイズに合わせて拡大縮小します。これは、SVGHEIGHT=および SVGWIDTH=システムオプションのデフォルト値がなく、100%の値を指定した場合と同じ処理が実行されるためです。この 2 つのシステムオプションで 100%を指定すると、SVG ドキュメントがビューポートの 100%に合わせて拡大縮小します。

embed タグで height 属性と width 属性を指定しない場合、ビューポートの寸法はブラウザによって決定されます。埋め込まれたドキュメントが期待通りには表示されない場合があります。

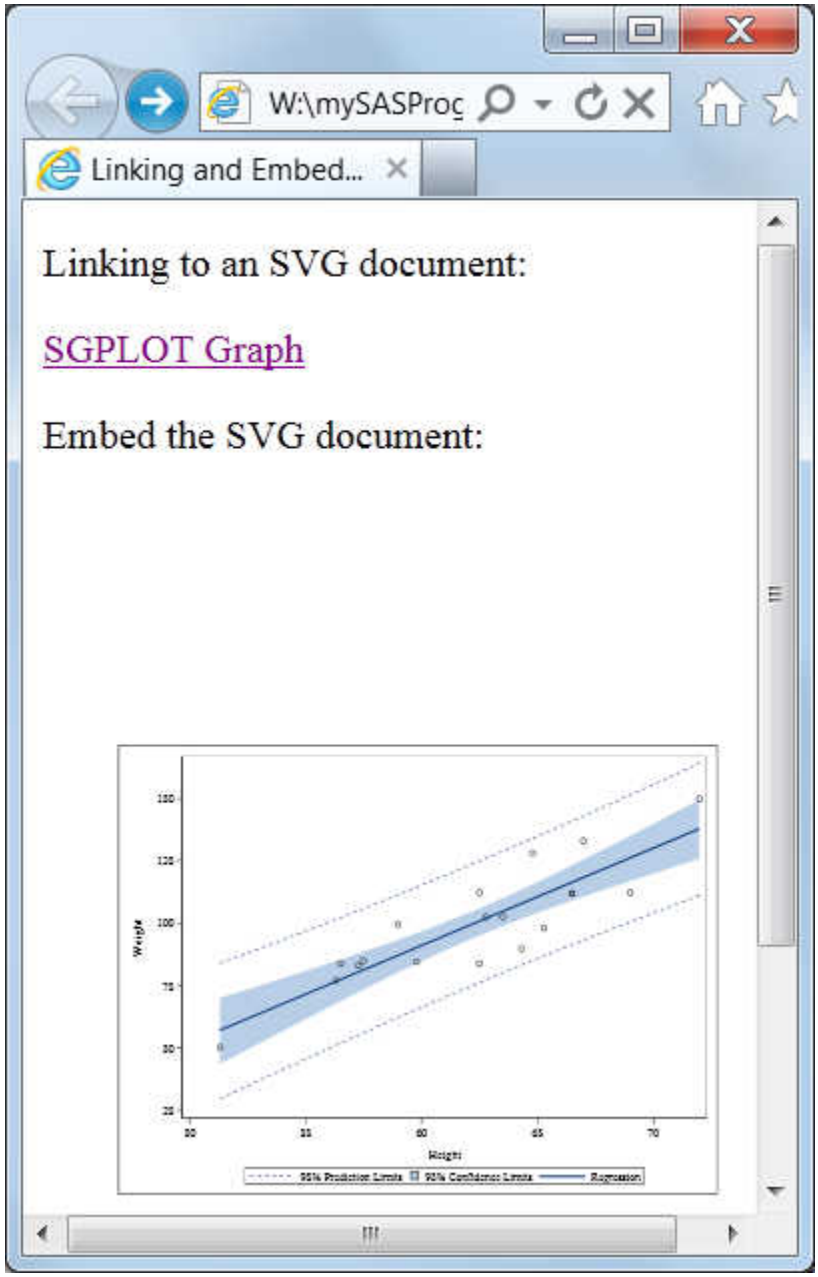
次の HTML ファイルは、スタンドアロン SVG ドキュメントの HTML ファイル内でのリンクおよび埋め込みを示しています。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
   <title>Linking and Embedding an SVG Document in an HTML Document</title>
   <meta http-equiv="X-UA-Compatible" content="IE=9".

</head>
<body>
<p>Linking to an SVG document:</p>
<a href="sasprt.svg">SGPlot Graph</a>
<p>Embed the SVG document:</p>
<embed src="sasprt.svg" type="image/svg+xml" height="400" width="300">
</body>
</html>
```

ここに HTML ファイルを示します。

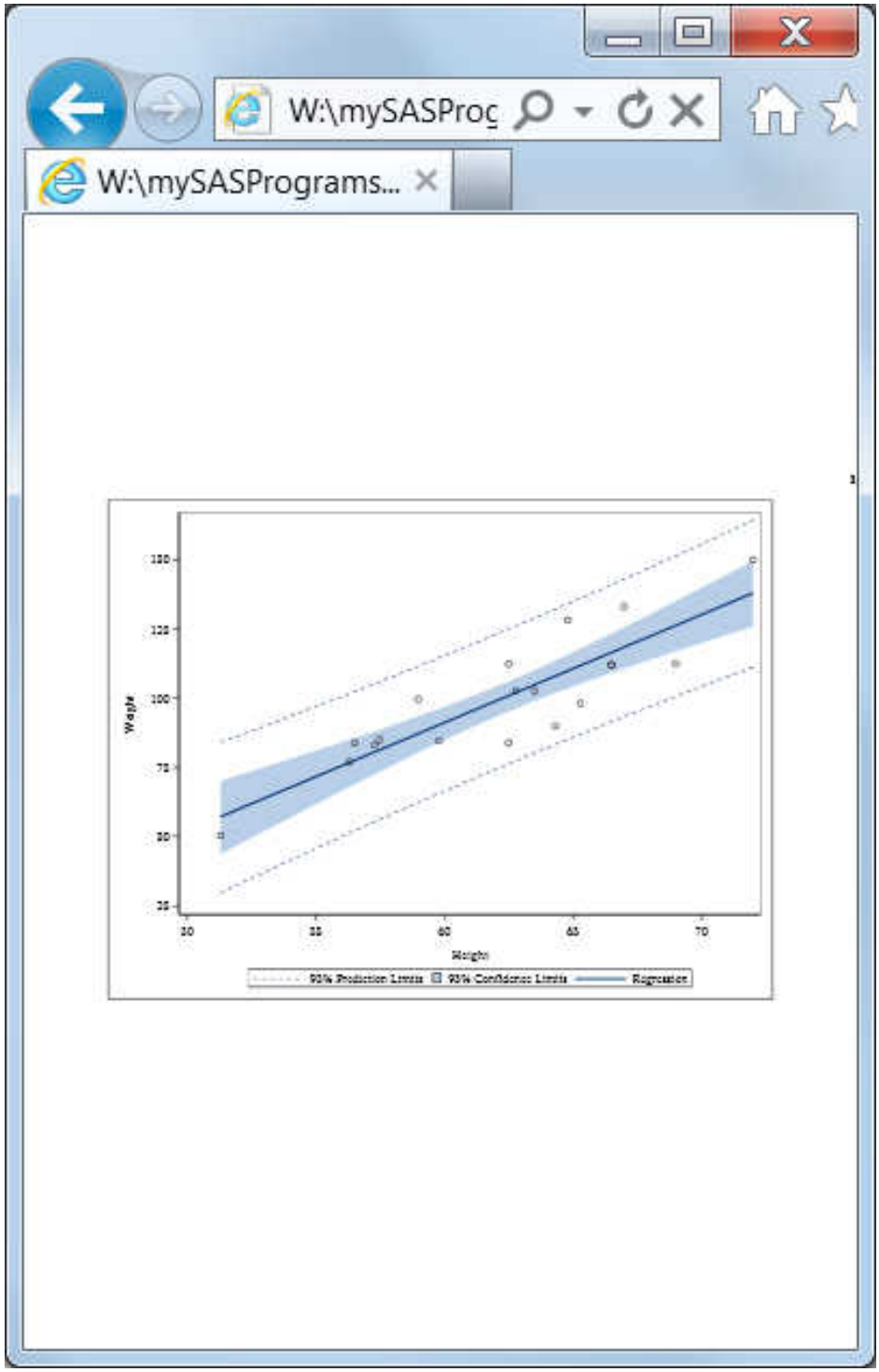
図 15.41 スタンドアロン SVG ドキュメントへのリンクと埋め込み SVG ドキュメントを表示する HTML ドキュメント



ビューポートは、高さ 400 ピクセル、幅 300 ピクセルです。デフォルト SVG システムオプション値を使用しているので、SVG ドキュメントはビューポートの 100%に合わせて拡大縮小します。

SGPLOT Graph リンクをクリックすると、次の SVG ドキュメントが表示されます。

図 15.42 HTML リンクをクリックして表示されるスタンドアロン SVG ドキュメント



ビューポートは、表示可能なブラウザウィンドウの領域で、SVG ドキュメントはビューポートの 100%に合わせて拡大縮小します。

次の例では、ODS HTML5 出力先を使用して、HTML ファイルに SVG グラフを埋め込みます。

```
ods html close;
ods html5 options(svg_mode="embed");
```

```
ods graphics /imagefmt=svg;
proc sgplot data=sashelp.stocks
  (where=(date >= "01jan2000"d and stock = "IBM"));
  title "Stock Trend";
  series x=date y=close;
  series x=date y=low;
  series x=date y=high;
run;
ods html5 close;
ods html;
```

HTML5 出力先のデフォルト svg_mode は INLINE です。SVG グラフを埋め込むためには、SVG_MODE="EMBED"を ODS HTML5 ステートメントのオプションとして指定する必要があります。HTML ファイル内の<EMBED>要素は次のとおりです。

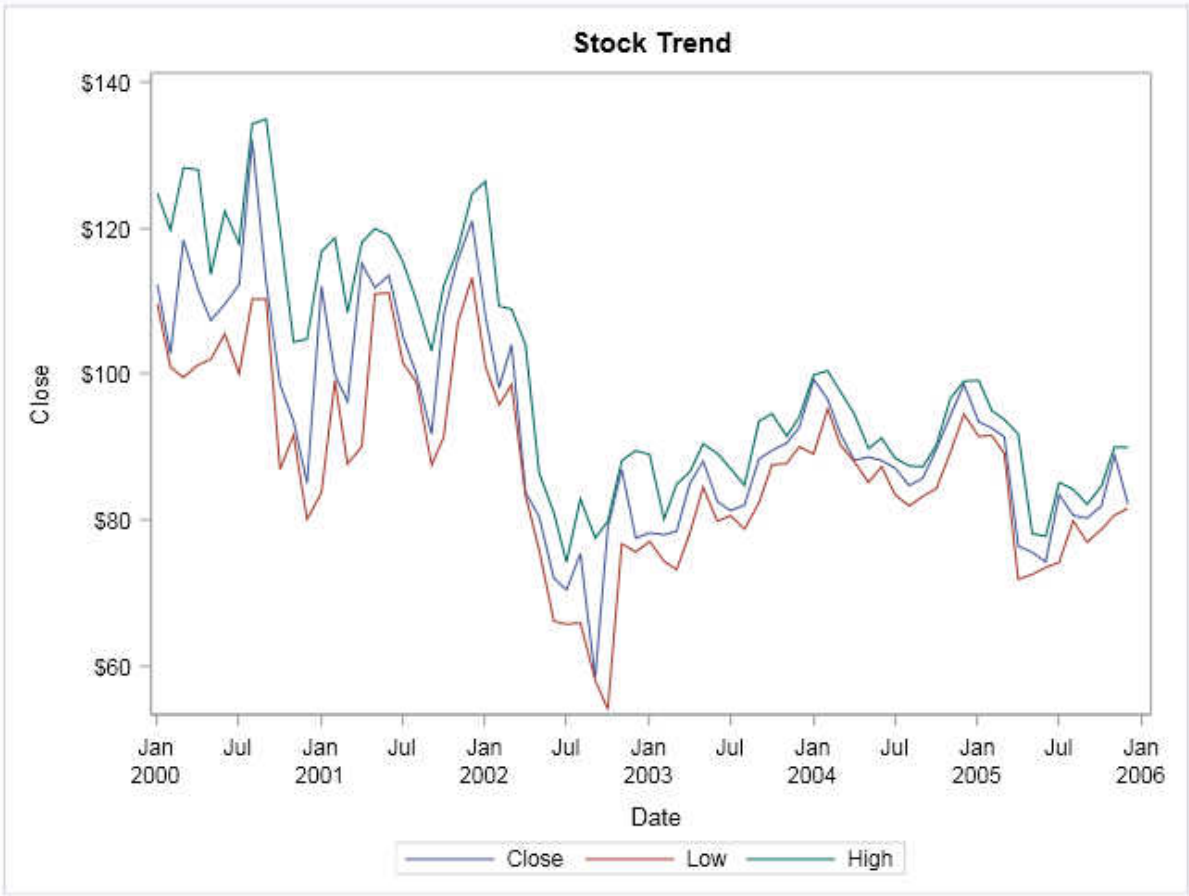
```
<embed style="height: 480px; width: 640px" src="SGPLOT.svg" type="image/svg+xml"/>
```

ODS Graphics SVG グラフを HTML5 ファイルに統合

ODS Graphics SVG グラフを HTML ファイル内に統合するには、ODS HTML5 ステートメントでオプション SVG_MODE='INLINE'を指定します。

```
ods html close;
ods html5 options(svg_mode="inline");
ods graphics /imagefmt=svg;
proc sgplot data=sashelp.stocks
  (where=(date >= "01jan2000"d and stock = "IBM"));
  title "Stock Trend";
  series x=date y=close;
  series x=date y=low;
  series x=date y=high;
run;
ods html5 close;
ods html;
```

ここに、SVG グラフが統合された HTML ファイルを示します。



ブラウザからの SVG ドキュメントの印刷

SVG ドキュメントの印刷は、ブラウザで制御されます。ブラウザウィンドウに表示されているものだけが印刷されます。

ユニバーサル印刷を使用した TIFF 画像の作成

SAS における TIFF 画像の処理

TIFF (Tagged image file format)画像は、文書処理、スキャン、ファックス、その他のアプリケーションで一般的に使用されるラスタ画像です。

SAS は、TIFF 6.0 の RGBA カラーおよび CMYK カラーをサポートしています。SAS は TIFF 画像を作成し、それを圧縮します。NOUPRINTCOMPRESSION システムオプションが設定されている場合は、SAS が作成する TIFF 画像は非常に大きくなります。

TIFF プリンタの説明を取得するには、SAS レジストリでプリンタを表示するか、次の QDEVICE プロシジャをサブミットして、SAS ログに出力を表示します。

```
proc qdevice;
   printer tiff;
run;
```

関連項目:
“ユニバーサルプリンタのカラーサポート” (234 ページ)

TIFF ユニバーサルプリンタ

SAS では、これらのユニバーサルプリンタを使用できます。

プリンタ名	説明
TIFF	RGBA カラーと透過性を使用して TIFF 画像を作成します。
TIFFk	CMYK カラーを使用して TIFF 画像を作成します。透過性はサポートされません。

TIFF プリンタは複数ページドキュメントをサポートしていません。プロシジャが複数ページを作成する場合や、複数のプロシジャが ODS PRINTER 出力のコード内で使用される場合には、最初のページのみが表示可能です。

TIFF 画像の作成

ODS PRINTER ステートメントを使用すると、TIFF 画像を作成できます。TIFF ユニバーサルプリンタを、PRINTERPATH=システムオプションの値か、ODS PRINTER ステートメントの PRINTER=オプションの値として指定します。

次に、TIFF 画像を作成するためのサンプルコードを示します。

```
ods html close;
ods printer printer=tiff;

...more SAS code...

ods printer close;
ods html;
```

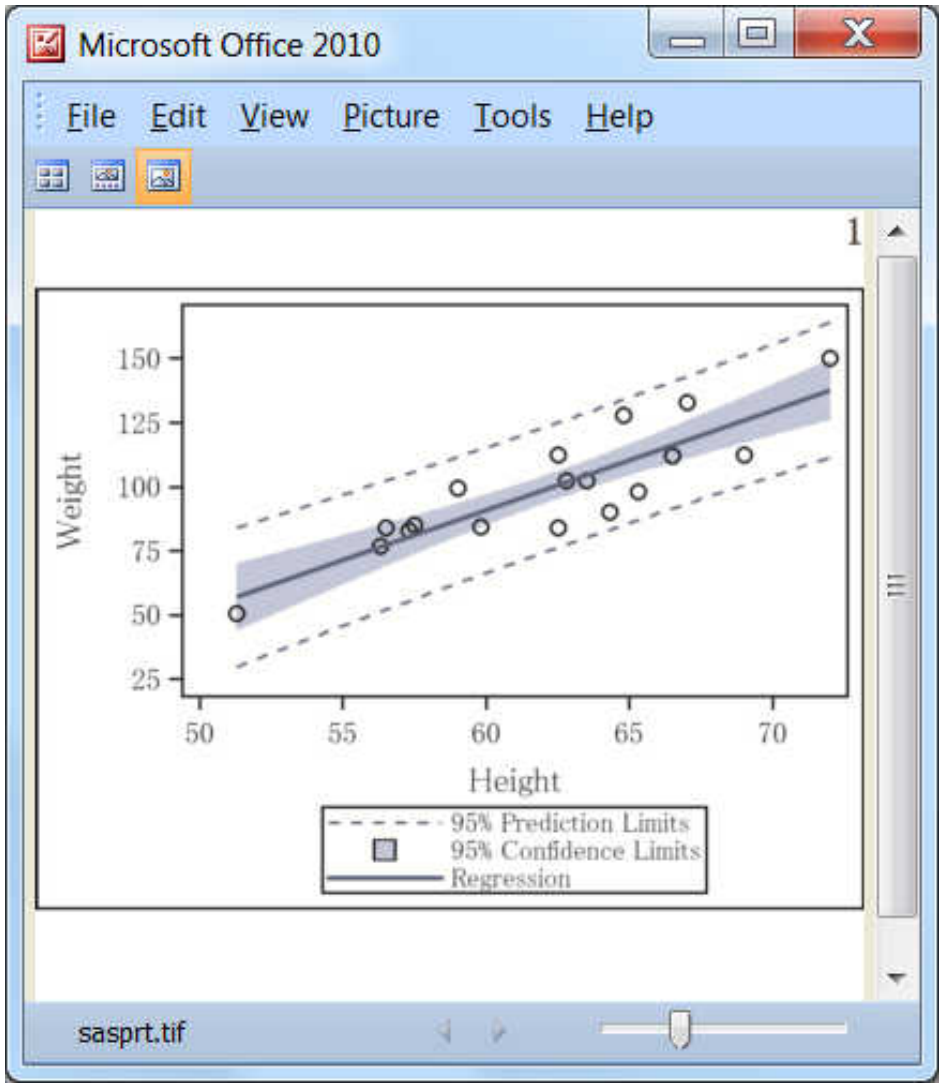
現在のディレクトリに、sasprt.tif ファイルが作成されます。

SAS/GRAPH では、TIFF デバイスは TIFF ユニバーサルプリンタへのショートカットです。SAS/GRAPH デバイスを使用した TIFF 画像を作成する詳細については、*SAS/GRAPH: Reference* を参照してください。

ODS PRINTER ステートメントを用いた TIFF 画像の作成例

例のデータセット Sashelp.Class および SGPLOT プロシジャを使用すると、次の ODS PRINTER ステートメントは、TIFF ファイル sasprt.tif を現在のディレクトリに出力します。

```
options printerpath=tiff papersize=("4in" "4in") nodate;
ods html close;
ods printer;
proc sgplot data=sashelp.class;
  reg x=height y=weight / CLM CLI;
run;
ods printer close;
ods html;
```



アニメーション GIF 画像と SVG ドキュメントの作成

アニメーション GIF 画像と SVG ドキュメントについて

複数ページ GIF 画像または SVG ドキュメントを ODS PRINTER 出力先を使用して作成する場合、SAS システムオプションの設定によって作成される GIF 画像または SVG ドキュメントをアニメーションにすることができます。GIF 画像または SVG ドキュメントのページごとに、出力ファイルに 1 つのフレームが作成されます。システムオプションでは、これらのアニメーション属性を構成できます。

- アニメーションファイルの作成を開始/停止します。
- フレームが表示される時間
- フレームが重ね合わせられるか、順番に実行されるか
- アニメーションループが繰り返される回数

- Web ページにドキュメントがロードされるとき、アニメーションをすぐに開始するかどうか(SVG ドキュメントのみ)
- フレームがフェードイン/フェードアウト表示をするかどうか、フェードイン/フェードアウトする場合はその間にフレームが重ね合わせられる、または順番に実行されるかどうか(SVG ドキュメントのみ)

SAS/GRAPH は、ODS HTML5、ODS HTML、および ODS LISTING 出力先のアニメーションファイルを作成するために必要です。詳細については、*SAS/GRAPH: Reference* を参照してください。

ODS PRINTER 出力先を開く前に、OPTIONS ステートメントを使用してオプションを設定します。

```
options printerpath=gif animation=start animduration=5 animloop=yes noanimoverlay;
ods printer file='myfile.gif';
```

この OPTIONS ステートメントでは、ANIMATION オプションがアニメーションファイルの作成を開始し、ANIMDURATION オプションは各フレームが 5 秒間保持されることを示します。ANIMLOOP オプションは、アニメーションループを連続して繰り返すために指定します。NOANIMOVERLAY オプションは、各フレームが順番に実行されることを示します。

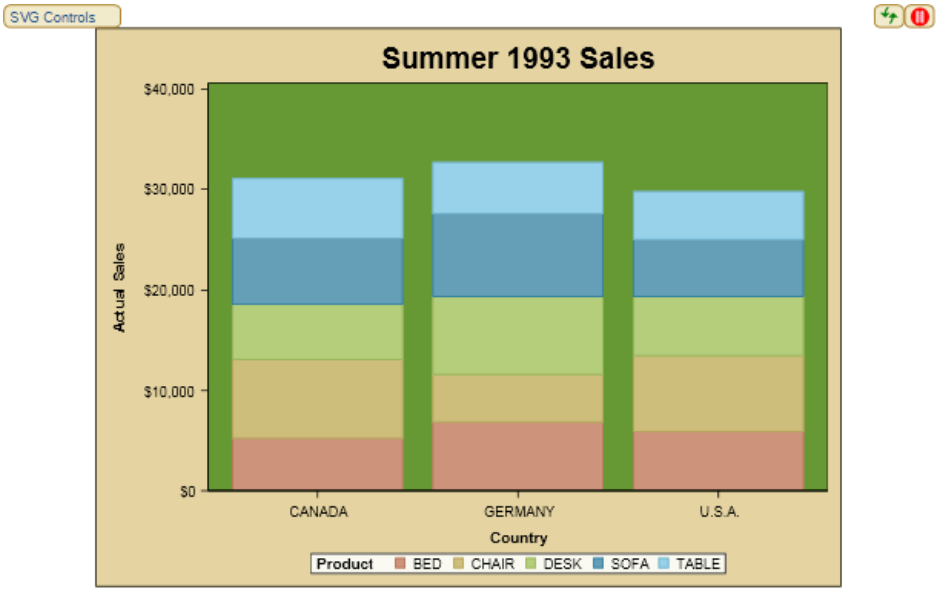
PRINTERPATH=オプションが SVG に設定されている場合、SVG アニメーションオプションを使用して、フェードイン属性とフェードアウト属性、および自動再生属性を構成できます。SVG で始まるアニメーションオプションは GIF 画像には影響を与えません。

オプションを設定し、PRINTER 出力先を開いた後で、次に SAS コードを実行してファイルに各フレームを作成します。SAS プロシジャが実行され、アニメーションフレームが作成されます。アニメーションオプションをプロシジャとプロシジャの間に使用して、フレームの表示時間とフェードイン/フェードアウト時間を変更できます。たとえば、特定のフレームの表示時間をより長く保持できます。OPTIONS ANIMDURATION=ステートメントをプロシジャの前に使用すれば、そのフレームの表示時間を増やせます。ANIMATION=STOP の指定で、アニメーションファイルの作成は終了します。ODS PRINTER CLOSE ステートメントを使用してファイルを閉じます。

ヒント アニメーションファイルにフレームを作成した後は、必ず ANIMATION=STOP を指定します。ANIMATION=START の設定が残っている場合、後続のプロシジャステートメントに関して、意図していないアニメーションファイルが作成される場合があります。

ファイルを Web ページに埋め込む、または Web ページからファイルへのリンクを作成するには、“HTML ファイル内の SVG ドキュメント” (322 ページ)を参照してください。

アニメーションファイルをブラウザで表示する場合、アニメーションコントロールボタンを使用すると、アニメーションのリセット()、アニメーションの一時停止()、およびアニメーションの再生()が行えます。**SVG コントロール**を切り替えて、コントロールボタンを表示したり非表示にしたりできます。コントロールボタンの付いたアニメーション SVG ドキュメントのフレームをここに示します。



iPad で表示する SVG ファイルを作成する場合、最適なサイジングのために SVGVIEW ユニバーサルプリンタを使用するのがベストプラクティスです。

アニメーション SVG ファイルを表示するには、Internet Explorer 9 またはそれ以降のバーションを使用します。アニメーション GIF ファイルは、Internet Explorer 9 より前のバーションでも表示できます。

アニメーションシステムオプション

アニメーションシステムオプションを使用して、アニメーション GIF や SVG ファイルの属性を構成します。すべてのシステムオプションが SVG ドキュメント用に構成できます。SVG で始まるオプションは GIF 画像には適用されません。アニメーションシステムオプションをここに示します。

表 15.22 *アニメーションシステムオプションと有効なユニバーサルプリンタ*

説明	オプション名	有効なユニバーサルプリンタ
アニメーションファイルの作成を開始/停止します。	ANIMATION	GIF および SVG
アニメーションの各フレームが表示される時間を指定します。	ANIMDURATION=	GIF および SVG

説明	オプション名	有効なユニバーサルプリンタ
アニメーションループが連続再生なのか一度だけの再生なのかを指定するか、または特定のアニメーションループ繰り返し回数を指定します。 ANIMLOOP=YES は連続ループを設定します。ANIMLOOP=NO は 1 ループで完了します。ANIMLOOP=*n* は、特定のループ回数を指定します。 SVG ドキュメントの場合は、ANIMLOOP=YES および ANIMLOOP=NO を使用します。ANIMLOOP=*n* (ここでは $n > 0$)を設定した場合、SVG ドキュメントは 1 ループで完了します。	ANIMLOOP=	GIF および SVG
アニメーションフレームが重ね合わせられるか、または順番に再生されるかを指定します。 フレームを重ね合わせる場合、出力が上書きされたと見られないためには、フレームにはあるレベルの透過性が必要です。	ANIMOVERLAY	GIF および SVG
SVG アニメーションが Web ブラウザですぐに開始するかどうかを指定します。 NOSVGAUTOPLAY を指定した場合、アニメーションは  をクリックすると開始します。	SVGAUTOPLAY	SVG
SVG フレームがフェードインするまでの秒数を指定します。	SVGFADEIN=	SVG
SVG フレームがフェードイン/アウトする際、フレームが前のフレームとオーバーラップするか、各フレームが順番に実行されるかを指定します。	SVGFADEMODE=	SVG
SVG フレームがフェードアウトするまでの秒数を指定します。	SVGFADEOUT=	SVG

例: アニメーション SVG ドキュメントの作成

データセット sashelp.prdsale には、1993 年と 1994 年のオフィスおよび家具の売上高データが含まれています。この例では、SGPLOT プロシジャを使用して、カナダ、ドイツ、アメリカのオフィスおよび家具製品の実際の売上数を縦棒にプロットして示します。縦棒にはそれぞれの国がプロットされます。縦棒ごとに製品別のデータが示されます。例では、それぞれの年の四半期毎にデータをグループ化し、8 つのチャートを作成しています。アニメーションを再生すると、各 SVG フレームが四半期の 1 つに関するチャートになります。チャートは季節ごとの色分けで表示され、製品の売上高の値が各縦棒内で変化します。この例のアニメーション再生を確認するには、support.sas.com を

参照してください。**Knowledge Base** の下で、**Samples & SAS Notes** を選択します。**`SVG animation`** を検索して、検索結果からプログラム seasons.sas を見つけます。

Create a data set for each quarter for the years 1993 and 1994. Each DATA step uses a WHERE clause to create a data set by year and quarter. The KEEP option in the SET statement specifies the variables that are in each of the data sets.

```
data work.q1y93 (where=(year=1993 and quarter=1));
set sashelp.prdsale(keep=Actual Country Product Quarter Year);
run;

data work.q2y93 (where=(year=1993 and quarter=2));
set sashelp.prdsale(keep=Actual Country Product Quarter Year);
run;

data work.q3y93 (where=(year=1993 and quarter=3));
set sashelp.prdsale(keep=Actual Country Product Quarter Year);
run;

data work.q4y93 (where=(year=1993 and quarter=4));
set sashelp.prdsale(keep=Actual Country Product Quarter Year);
run;

data work.q1y94 (where=(year=1994 and quarter=1));
set sashelp.prdsale(keep=Actual Country Product Quarter Year);
run;

data work.q2y94 (where=(year=1994 and quarter=2));
set sashelp.prdsale(keep=Actual Country Product Quarter Year);
run;

data work.q3y94 (where=(year=1994 and quarter=3));
set sashelp.prdsale(keep=Actual Country Product Quarter Year);
run;

data work.q4y94 (where=(year=1994 and quarter=4));
set sashelp.prdsale(keep=Actual Country Product Quarter Year);
run;
```

Create a style for each season. The four TEMPLATE procedures create a style for each season by specifying seasonal colors for the different parts of the chart. The colors for the vertical bars are not part of the style because they are automatically generated by the SGPLOT procedure.

```
proc template;
  define style winter;
  parent = Styles.meadow;
       style body from body;

     style GraphColors from GraphColors /
           "gborderlines" = cx000000
           "greferencelines" = cx000000
           "gaxis" = cx000000
           "gwalls" = cx83838C
           ;
  style GraphBackground /
      Color = cxB3B2BF
```

```
 ;
end;
quit;


proc template;
  define style spring;
  parent = Styles.meadow;
       style body from body;

  style GraphColors from GraphColors /
        "gborderlines" = cx000000
        "greferencelines" = cx000000
        "gaxis" = cx000000
        "gwalls" = cxFF9999
        ;
  style GraphBackground from GraphBackground "Graph background attributes" /
      Color = cxFFFF99
  ;
end;
quit;


proc template;
  define style summer;
  parent = Styles.meadow;
       style body from body;

  style GraphColors from GraphColors /
        "gborderlines" = cx000000
        "greferencelines" = cx000000
        "gaxis" = cx000000
        "gwalls" = cx669933
        ;
  style GraphBackground from GraphBackground "Graph background attributes" /
      Color = cxE5D4A1
  ;
end;
quit;

proc template;
  define style fall;
  parent = Styles.meadow;
       style body from body;

  style GraphColors from GraphColors /
        "gborderlines" = cx000000
        "greferencelines" = cx000000
        "gaxis" = cx000000
        "gwalls" = cx996633
        ;
  style GraphBackground from GraphBackground "Graph background attributes" /
      Color = cxD9A465
  ;
end;
quit;
```

```
ODS LISTING CLOSE;
```

Set the options to create an animated file for an SVG document. Set the PRINTERPATH= option to create an SVG document. The ANIMATION= option starts creating the animation. Each page in the animation is held in view for 3 seconds as specified in the ANIMDURATION option. The SVGFADEIN=0 and SVGFADEOUT=0 options specify that the pages do not fade in or out of view. The NOANIMOVERLAY option specifies that the pages are played sequentially. The ODS PRINTER CLOSE statement closes any files that are open for the PRINTER destination. Because the ANIMLOOP= option is not specified, the default of YES is used and the animations loop continuously.

```
goptions reset=all;
options printerpath=svg animate=start animduration=3 svgfadein=0 svgfadeout=0
noanimoverlay nodate nonumber;

ODS PRINTER CLOSE;
```

Create an SVG document for each quarter, using the SGPLOT procedure. The %LET macro variable is used to name the SVG file. The first ODS PRINTER statement opens the PRINTER destination and creates the SVG file. After the first ODS PRINTER statement, an ODS PRINTER statement is used before each procedure to specify the style that indicates the seasonal colors to use to create a chart. The TITLE statement specifies the season and the year that is reported. Each SGPLOT procedure plots the sales for each country by using identical VBAR and YAXIS options in each procedure. The **`vbar country / response=actual group=product;`** statement specifies to create a vertical bar for each country. Each vertical bar contains sales data for each product. The visual aspects for each product in vertical bar are automatically determined by the SGPLOT procedure. The YAXIS statement specifies the values to plot for the Y axis.

```
%let name=seasons;

ODS PRINTER file="&name..svg";
ods printer style=winter;
title1 h=18pt "Winter 1993 Sales";
proc sgplot data=work.q1y93 uniform=all;
vbar country / response=actual group=product;
yaxis values=(0 to 40000 by 10000);
run;

ods printer style=spring;
title1 h=18pt "Spring 1993 Sales";
proc sgplot data=work.q2y93 uniform=all;
vbar country / response=actual group=product;
yaxis values=(0 to 40000 by 10000);
run;


ods printer style=summer;
title1 h=18pt "Summer 1993 Sales";
proc sgplot data=work.q3y93 uniform=all;
vbar country / response=actual group=product;
yaxis values=(0 to 40000 by 10000);
run;
```

```
ods printer style=fall;
title1 h=18pt "Fall 1993 Sales";
proc sgplot data=work.q4y93 uniform=all;
vbar country / response=actual group=product ;
yaxis values=(0 to 40000 by 10000);
run;

ods printer style=winter;
title1 h=18pt "Winter 1994 Sales";
proc sgplot data=work.q1y94 uniform=all;
vbar country / response=actual group=product ;
yaxis values=(0 to 40000 by 10000);
run;

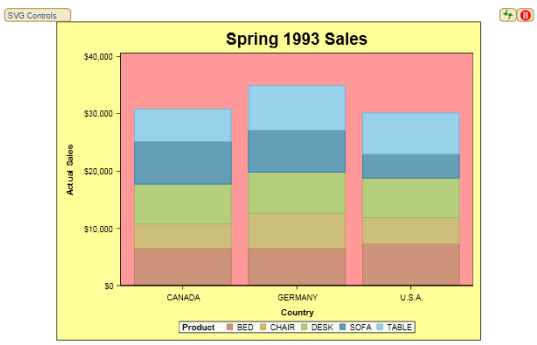
ods printer style=spring;
title1 h=18pt "Spring 1994 Sales";
proc sgplot data=work.q2y94 uniform=all;
vbar country / response=actual group=product ;
yaxis values=(0 to 40000 by 10000);
run;

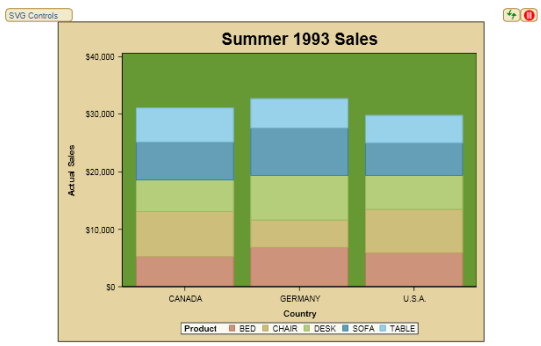
ods printer style=summer;
title1 h=18pt "Summer 1994 Sales";
proc sgplot data=work.q3y94 uniform=all;
vbar country / response=actual group=product ;
yaxis values=(0 to 40000 by 10000);
run;

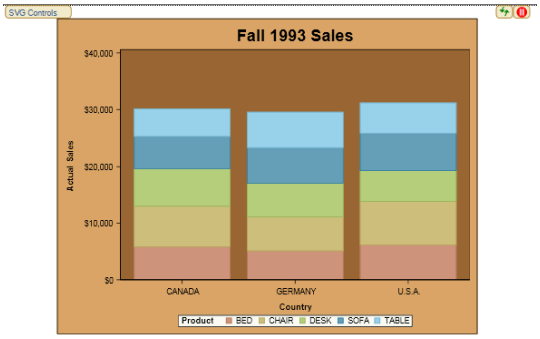
ods printer style=fall;
title1 h=18pt "Fall 1994 Sales";
proc sgplot data=work.q4y94 uniform=all;
vbar country / response=actual group=product ;
yaxis values=(0 to 40000 by 10000);
run;

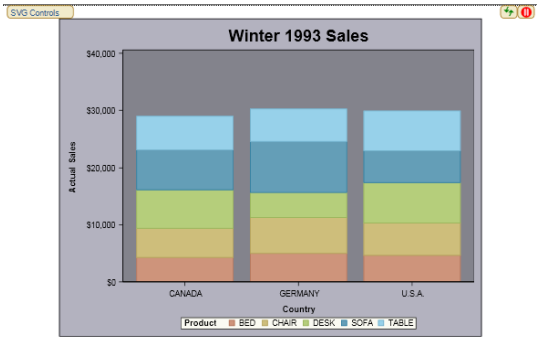
options animation=stop;
ods printer close;
```

ここに、1993 年の季節ごとに分けたグラフを示します。






SVG Controls
Fall 1993 Sales
$40,000
$30,000
$20,000
$10,000
$0
Actual Sales
CANADA
GERMANY
U.S.A.
Country
Product BED CHAIR DESK SOFA TABLE


SVG Controls
Winter 1993 Sales
$40,000
$30,000
$20,000
$10,000
$0
Actual Sales
CANADA
GERMANY
U.S.A.
Country
Product BED CHAIR DESK SOFA TABLE

2 部

ウィンドウ環境の概念

16 章

SAS ウィンドウ環境の紹介

SAS ウィンドウ環境について

SAS では、SAS を使いやすくするグラフィカルユーザーインターフェイスが用意されています。SAS のすべてのウィンドウをまとめて SAS ウィンドウ環境と呼びます。

SAS ウィンドウ環境には、SAS プログラムの作成に使用するウィンドウも含まれます。他に、コードを記述しなくても、データを操作したり、SAS 設定を変更したりできるウィンドウもあります。

SAS データセットを操作する場合、または SAS セッションの一部の側面を制御する場合は、SAS プログラムを記述する代わりに、SAS ウィンドウ環境を使用すると便利な場合があります。

SAS ウィンドウ環境のメインウィンドウ

SAS ウィンドウの概要

SAS ウィンドウには、どの動作環境でも同じように操作する複数の機能(メニュー、ツールバー、オンラインヘルプ)があります。ツールバー、アイコン、メニューなど、SAS ウィンドウ環境のさまざまな機能をカスタマイズできます。

SAS ウィンドウ環境の 5 つのメインウィンドウは、**エクスプローラ**、**結果**、**拡張エディタ**、**ログ**、**出力**ウィンドウです。

注: SAS ウィンドウの配置は、動作環境によって異なります。たとえば、Microsoft Windows の動作環境では、**拡張エディタ**ウィンドウが**プログラム**エディタの代わりに表示されます。

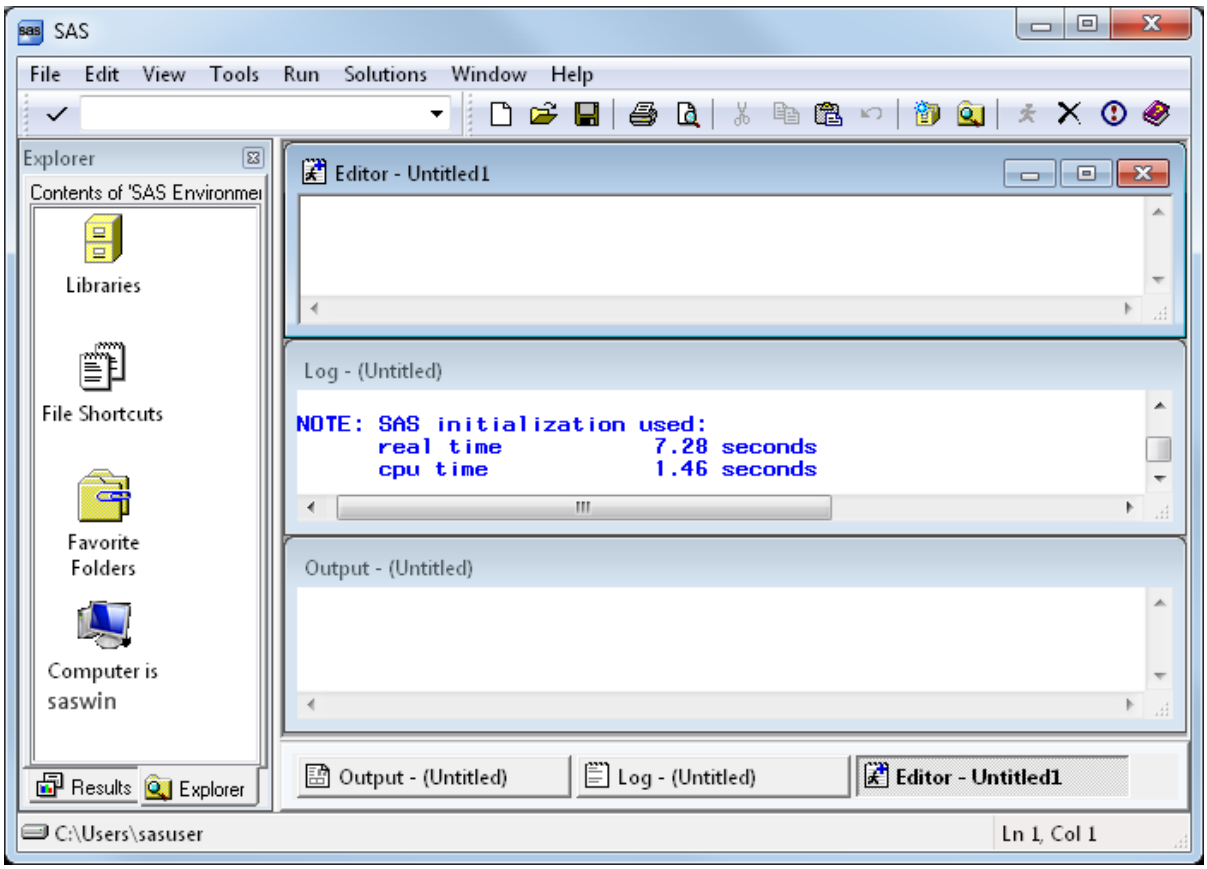
最初に SAS を起動すると、**拡張エディタ**、**ログ**、**出力**、**エクスプローラ**ウィンドウが表示されます。SAS プログラムを実行すると、デフォルト出力(HTML)が**結果**ウィンドウに表示されます。プログラムで PUT ステートメントを使用する場合、デフォルトで SAS **ログ**に出力が書き込まれます。

注: このセクションでは、Microsoft Windows 動作環境で例を作成しています。他の動作環境のメニューとツールバーも、同じような外観と動作になります。

Windows 固有
Microsoft Windows を使用している場合、アクティブなウィンドウによって、メインメニューバーで使用可能なメニュー項目が決まります。

次の表示画面は、SAS ウィンドウの配置の例を示します。**エクスプローラ**ウィンドウに、アクティブなライブラリが表示されます。

図 16.1 *SAS ウィンドウ環境でのウィンドウ*



SAS エクスプローラウィンドウ

SAS エクスプローラウィンドウの使用

エクスプローラウィンドウを使用すると、ウィンドウ環境でファイルを管理できます。SAS **エクスプローラ**を使用すると、次のタスクを実行できます。

- SAS ファイルのリストを表示します。
- 新しい SAS ファイルを作成します。
- ライブラリを表示、追加、削除します。
- 外部ファイルへのショートカットを作成します。
- SAS ファイルを開き、その内容を表示します。
- ファイルを移動、コピー、削除します。
- **ライブラリの作成**ウィンドウなどの関連ウィンドウを開きます。

SAS エクスプローラウィンドウを開く

SAS エクスプローラを開くには、次の方法があります。

コマンド:
コマンドラインに EXPLORER と入力し、Enter を押します。

メニュー:
表示 ⇨ **エクスプローラ**を選択します。

SAS エクスプローラの表示(ツリービュー付き/なし)

エクスプローラウィンドウは、ツリービュー付きまたはツリービューなしで表示できます。.**エクスプローラ**をツリービュー付きで表示すると、ファイルの階層を表示できます。ツリービューを表示するには、**表示**メニューから**ツリーを表示**を選択します。ツリービューをオフにするには、メニューで**ツリーを表示**の選択を解除します。

注: **エクスプローラ**ウィンドウのサイズを変更するには、ウィンドウの縁または角をドラッグします。**エクスプローラ**ウィンドウの左右のペインのサイズを変更するには、2 つのペインの間にあるスプリットバーをクリックして右または左に動かします。

拡張エディタウィンドウ

拡張エディタウィンドウの使用

拡張エディタウィンドウでは、SAS プログラムの入力、編集、サブミット、保存が行えます。

拡張エディタウィンドウを開く

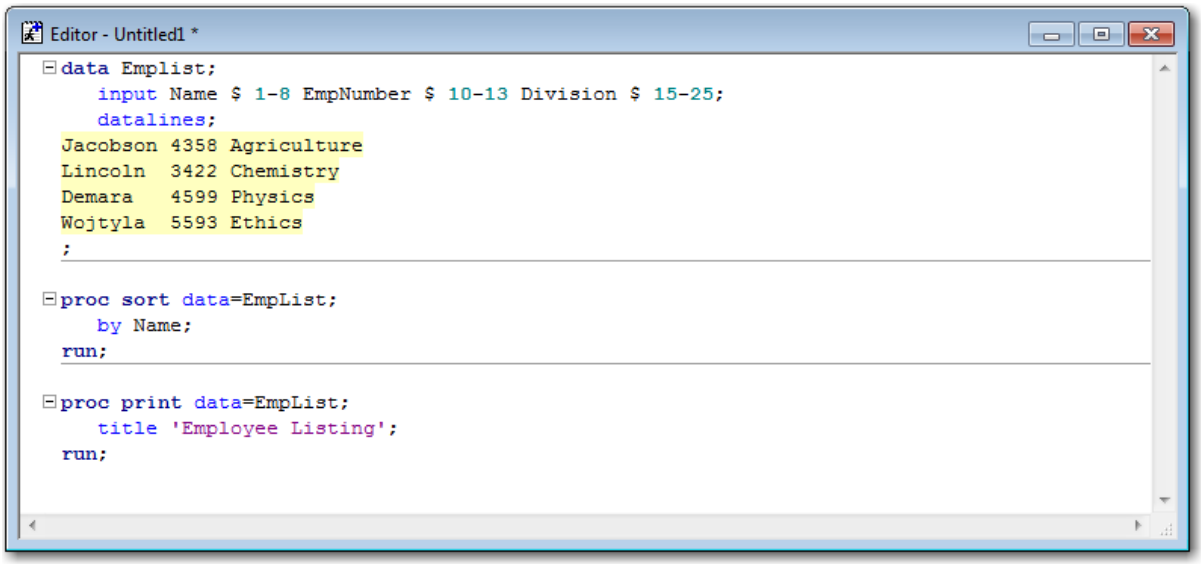
拡張エディタウィンドウを開くには、メインメニューから**表示** ⇨ **拡張エディタ**を選択します。

注: SAS ウィンドウ環境で SAS プログラムを開くには、それらのプログラムを**拡張エディタ**ウィンドウにドラッグアンドドロップします。

拡張エディタウィンドウでのプログラムの表示

次の例では、**拡張エディタ**ウィンドウに SAS プログラムを表示します。

図 16.2 *拡張エディタウィンドウの例*



注: Microsoft Windows の動作環境では、デフォルトで、**拡張エディタ**ウィンドウが**プログラムエディタ**ウィンドウの代わりに表示されます。**プログラムエディタ**ウィンドウを開くには、**拡張エディタ**ウィンドウを開く場合と同様の手順に従います、ただし、この場合は、メインメニューで**表示** ⇨ **プログラムエディタ**を選択します。または、コマンドラインに PROGRAM または PGM と入力し、Enter を押します。

ログウィンドウ

ログウィンドウの使用

ログウィンドウでは、SAS セッションや SAS プログラムに関するメッセージを表示できます。サブミットしたプログラムで予期しない結果が発生した場合、ログを参照すると、エラーの特定に役立ちます。PUT ステートメントを使用してプログラムの出力結果を**ログ**に書き出すこともできます。

注: ウィンドウを最大化したときにログの行を折り返さないようにするには、LINESIZE=システムオプションを使用します。

ログウィンドウを開く

ログウィンドウは、次の方法で開くことができます。

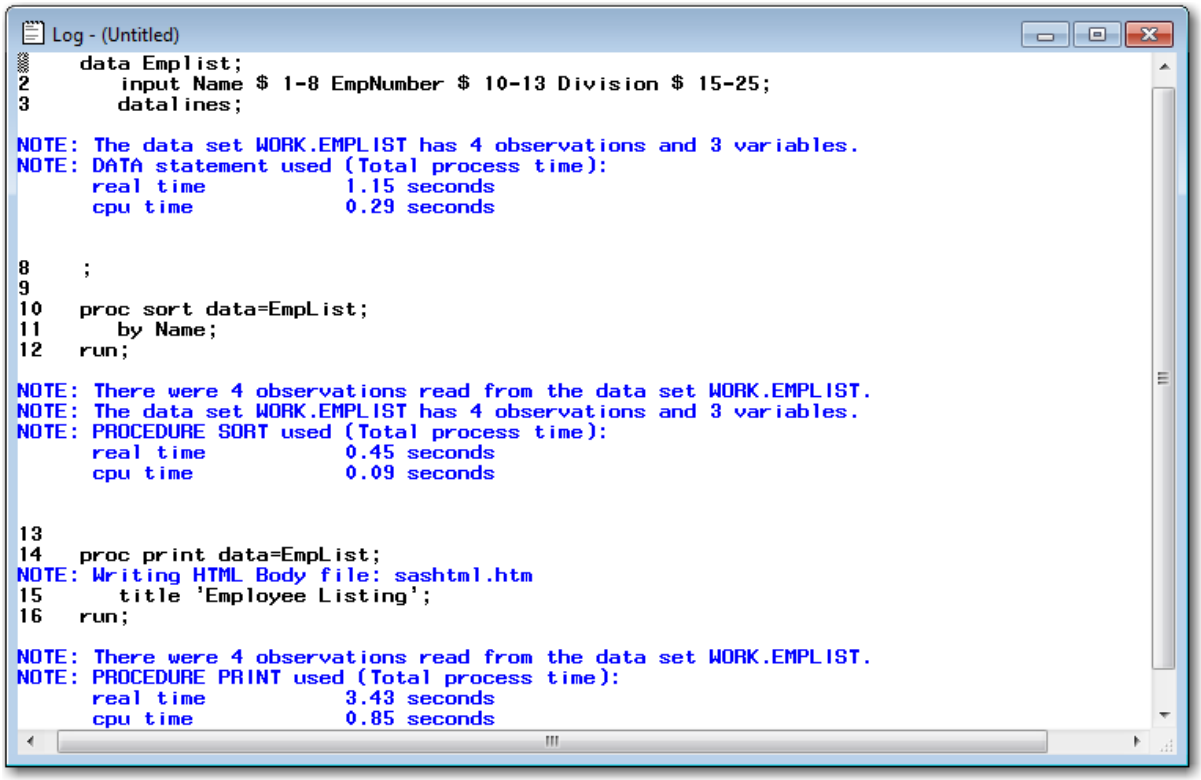
コマンド:
コマンドラインに LOG と入力し、Enter を押します。

メニュー:
表示 ⇨ **ログ**を選択します。

ログ出力の表示

次に、ログ出力の例を示します。

図 16.3 ログウィンドウの出力の例



結果ウィンドウ

結果ウィンドウの使用

結果ウィンドウでは、SAS プログラムから HTML 出力結果を表示できます。HTML はデフォルト出力タイプで、HTMLBlue はデフォルト出力スタイルです。**結果**ウィンドウは、ツリー構造を使用し、SAS を実行すると使用可能になる出力のさまざまなタイプをリストします。各ファイルを表示、保存、印刷できます。**結果**ウィンドウは、SAS プログラムを実行し、出力を生成するまで空です。SAS プログラムをサブミットすると、出力が**結果ビューア**に表示され、ファイルが**結果**ウィンドウにリストされます。

結果ウィンドウを開く

結果ウィンドウは、次の方法で開くことができます。

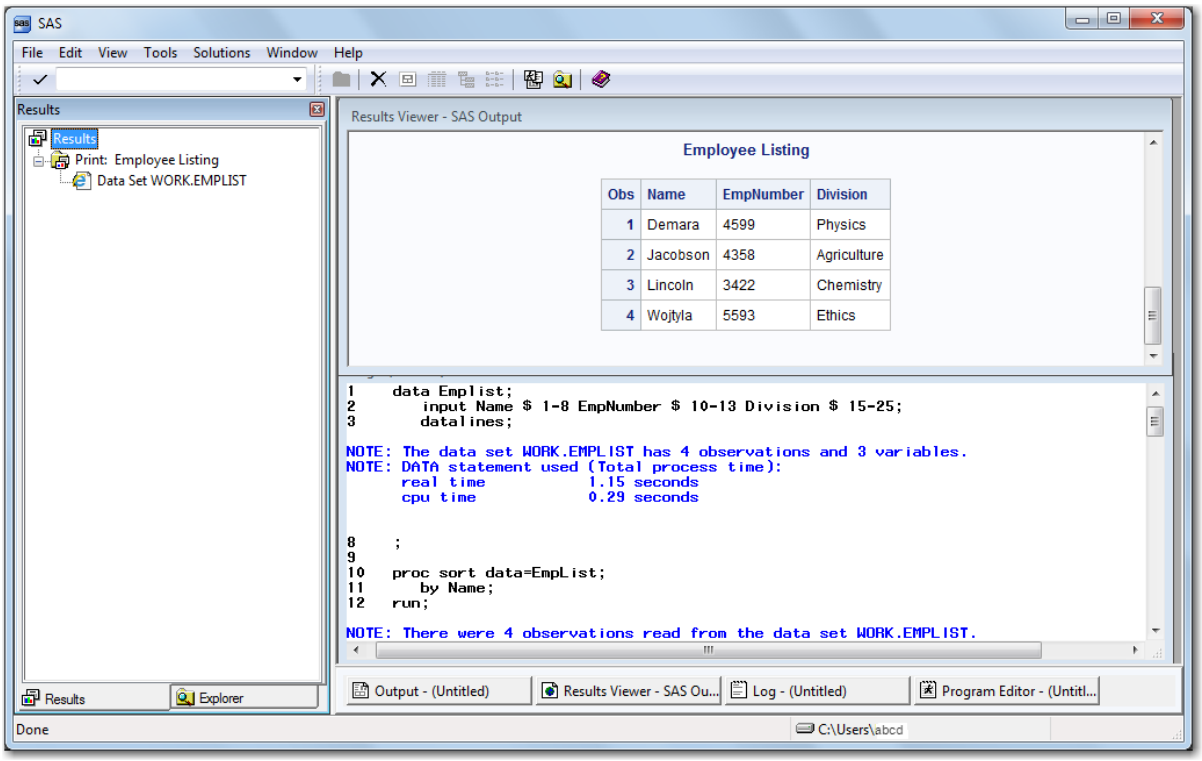
コマンド:
　コマンドラインに ODSRESULTS と入力し、Enter を押します。

メニュー:
　表示 ⇨ **結果**を選択します。

結果ウィンドウでの出力の表示

次の表示画面の左ペインに**結果**ウィンドウが表示され、右ペインに**結果ビューア**が表示されます。結果ビューアには、デフォルト HTML 出力が表示されます。SAS プログラムの実行時に作成されたファイルが**結果**ウィンドウに表示されます。

図 16.4 結果ウィンドウと結果ビューア



出力ウィンドウ

出力ウィンドウの使用

出力ウィンドウでは、SAS プログラムからリスト出力を表示できます。デフォルトでは、**出力**ウィンドウは他のウィンドウの後ろに配置されます。リスト出力を作成すると、**出力**ウィンドウがディスプレイの前面に自動的に移動します。

注: ウィンドウを最大化したときに出力の行を折り返さないようにするには、LINESIZE=システムオプションを使用します。

出力ウィンドウを開く

出力ウィンドウは、次の方法で開くことができます。

コマンド:

- コマンドラインに OUTPUT または OUT と入力し、Enter を押します。
- コマンドラインに LISTING または LST と入力し、Enter を押します。

メニュー:

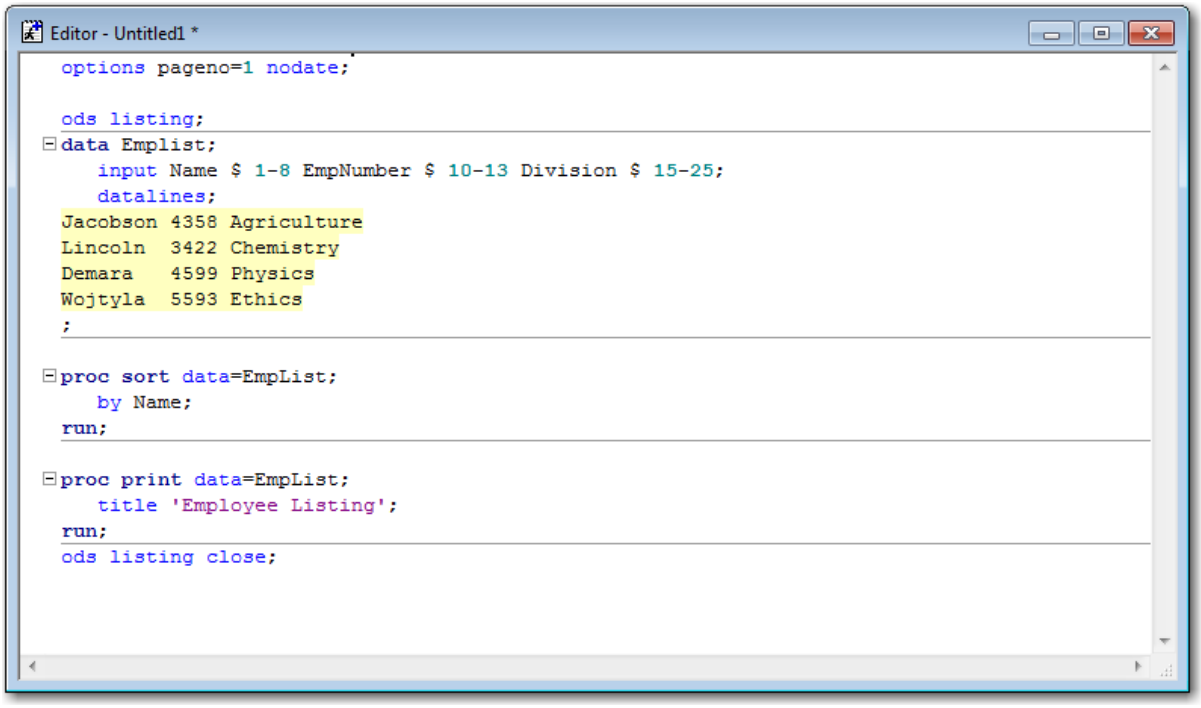
表示 ⇨ **出力**を選択します。

LISTING 出力の作成と表示

LISTING 出力はデフォルト出力タイプではないため、ODS ステートメントを使用して LISTING 出力先を開く必要があります。LISTING 出力に加え、HTML 出力も生成されます。

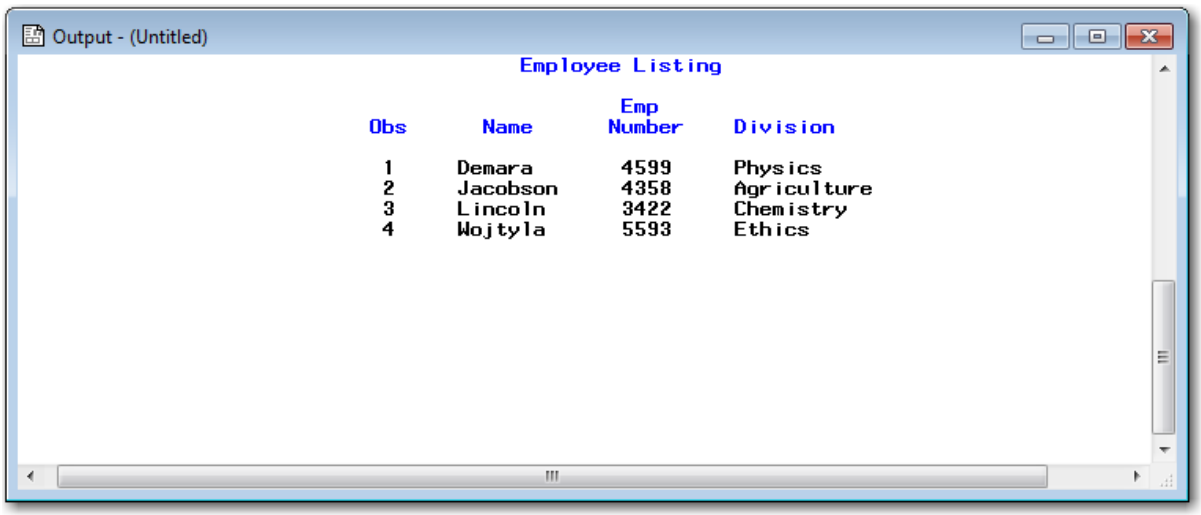
次の例では、LISTING 出力を生成するプログラムを示します。RUN ステートメントと DATA ステートメントの間に、ODS ステートメントがあります。

図 16.5 *リスト出力を生成するプログラムの例*



SAS は次の LISTING 出力を作成します。

図 16.6 *出力ウィンドウでのリスト出力の例*

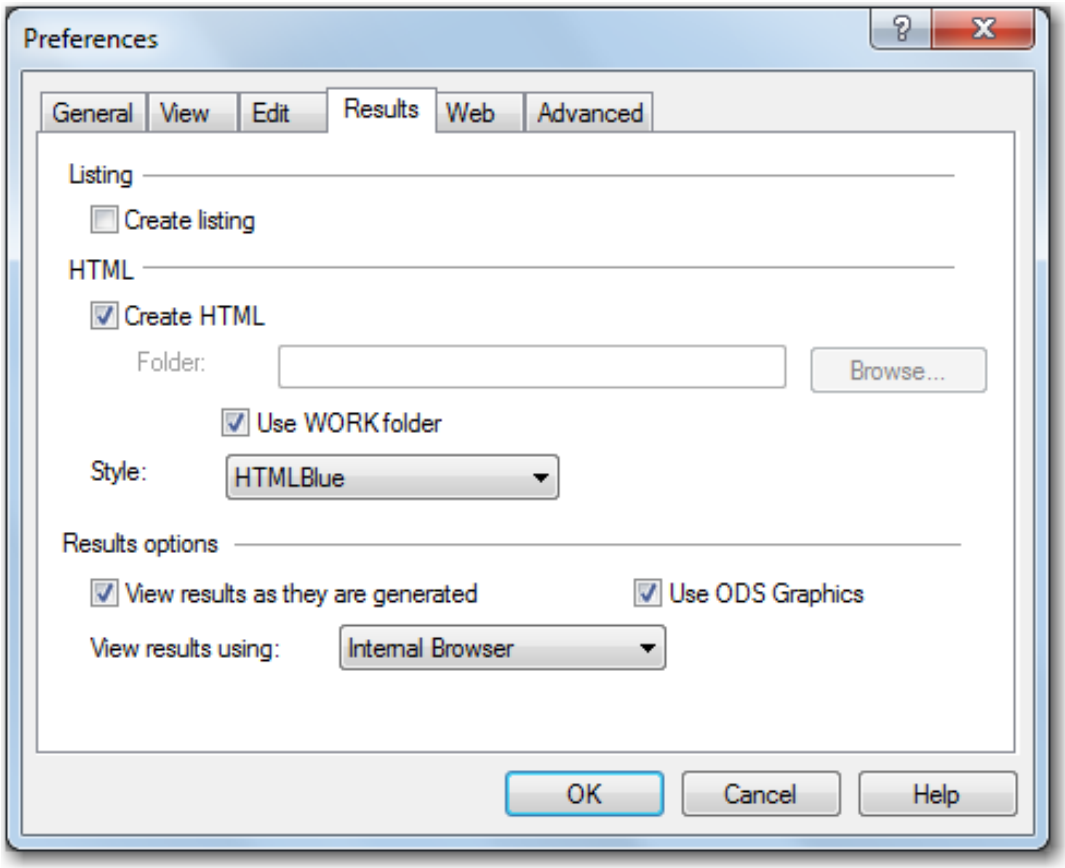


プリファレンスダイアログボックスによる出力タイプの選択

プリファレンスダイアログボックスを使用すると、出力タイプを選択し、システムプリファレンスを設定できます。**プリファレンス**ダイアログボックスの各タブは、項目の関連グループを保持します。**プリファレンス**ダイアログボックスにアクセスするには、**ツール** ⇨ **オプション** ⇨ **プリファレンス**を選択します。

次に、**プリファレンス**ダイアログボックスの例を示します。ここでは、**結果**タブが選択されています。

図 16.7 *プリファレンスダイアログボックスの例*



複数のデフォルト値が**結果**タブで選択されています。HTML では、**HTML を作成する**がデフォルト出力タイプ、**HTMLBlue** がデフォルト出力スタイルです。**ODS Graphics を使用する**もデフォルトで選択されています。**ODS Graphics を使用する**ボックスが選択されている場合、ODS グラフィックスをサポートするプロシジャを実行する際にグラフを自動的に作成できます。このボックスを選択または選択解除すると、SAS を呼び出すときに ODS グラフィックスのオンとオフを切り替えることができます。

リスト出力を生成するには、リストの下の**リストを作成する**ボックスを選択します。**HTML を作成する**の選択を解除し、**リストを作成する**ボックスを選択すると、リスト出力のみが生成されます。

SAS ウィンドウ環境でのナビゲーション

SAS ナビゲーションの概要

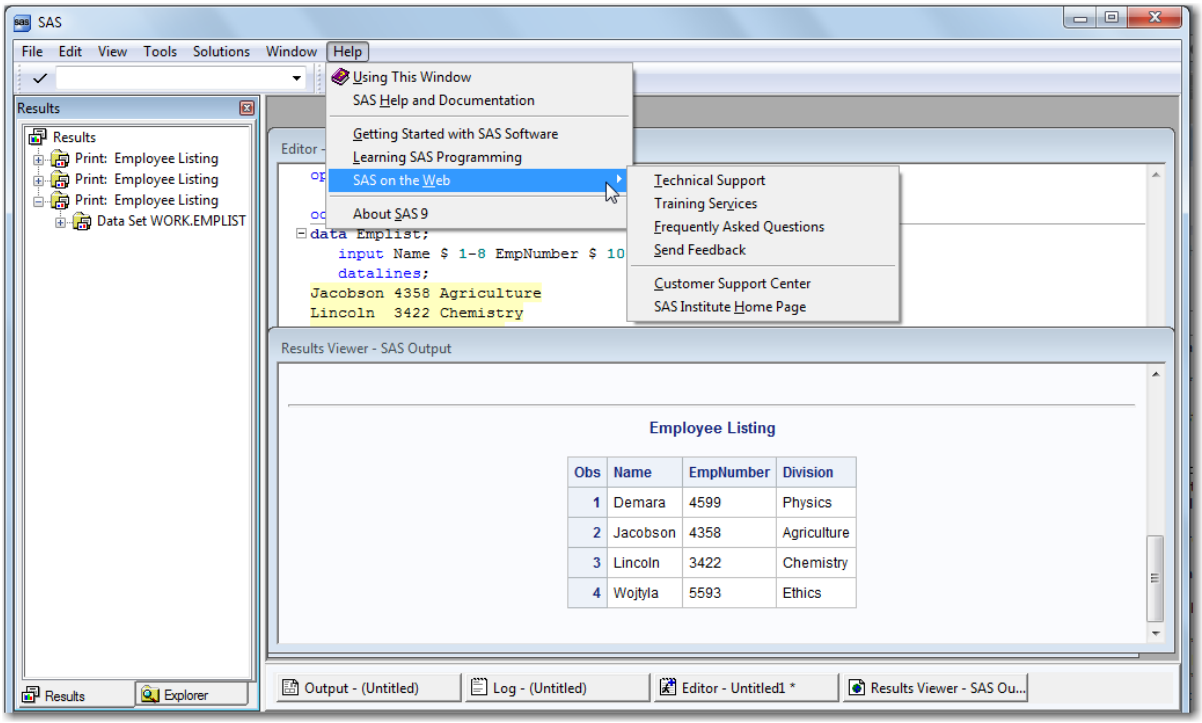
SAS ウィンドウには、どの動作環境でも同じように操作する複数の機能(メニュー、ツールバー、オンラインヘルプ)があります。これらの機能の多くは、メニューから**ツール** ⇨ **カスタマイズ**を選択してカスタマイズできます。これらの機能の詳細については、各動作環境に対応するドキュメントを参照してください。

SAS のメニュー

メニューには、選択可能なオプションのリストが含まれています。

次の例では、メニューバーから**ヘルプ**を選択したときに使用可能なメニューオプションが表示されます。

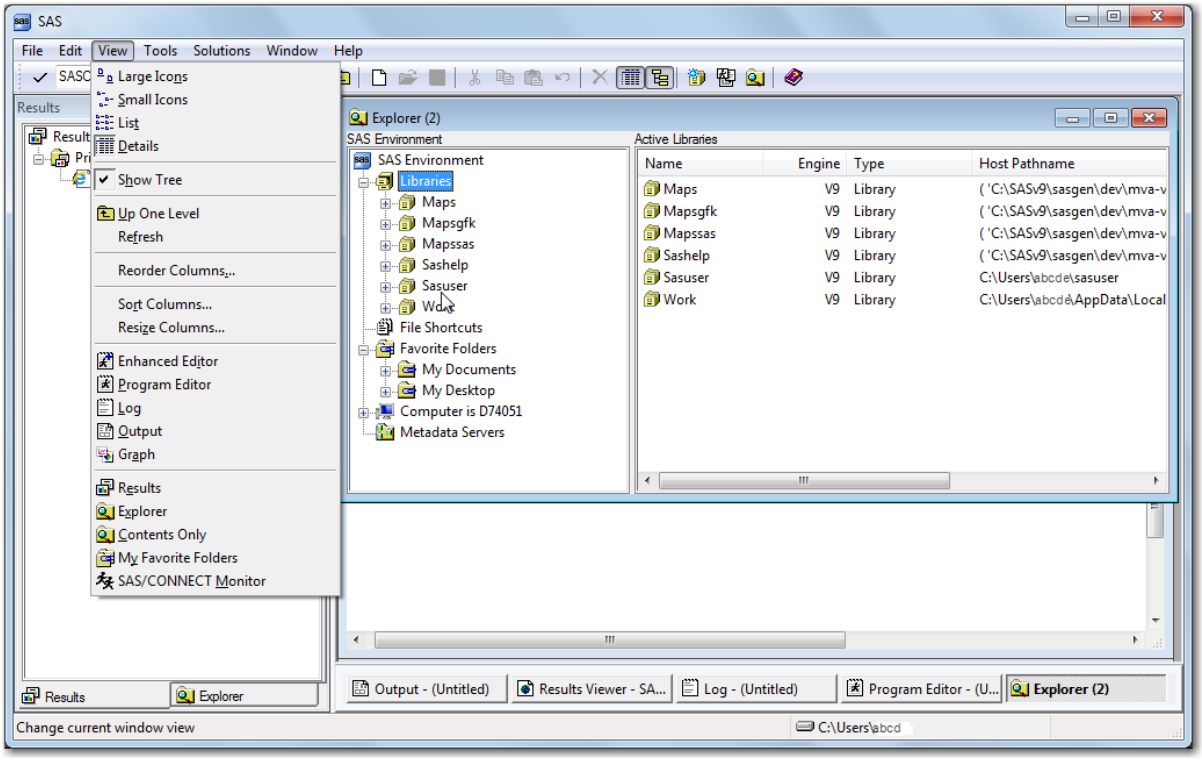
図 16.8 *ヘルプメニュー*



使用しているウィンドウが変わると、メニュー選択項目も変わります。たとえば、**表示**メニューから**エクスプローラ**を選択し、再び**表示**を選択すると、**エクスプローラ**ウィンドウがアクティブな場合に使用可能な**表示**オプションがメニューに表示されます。

次の表示画面では、**エクスプローラ**ウィンドウがアクティブになると、**表示**メニューが表示されます。

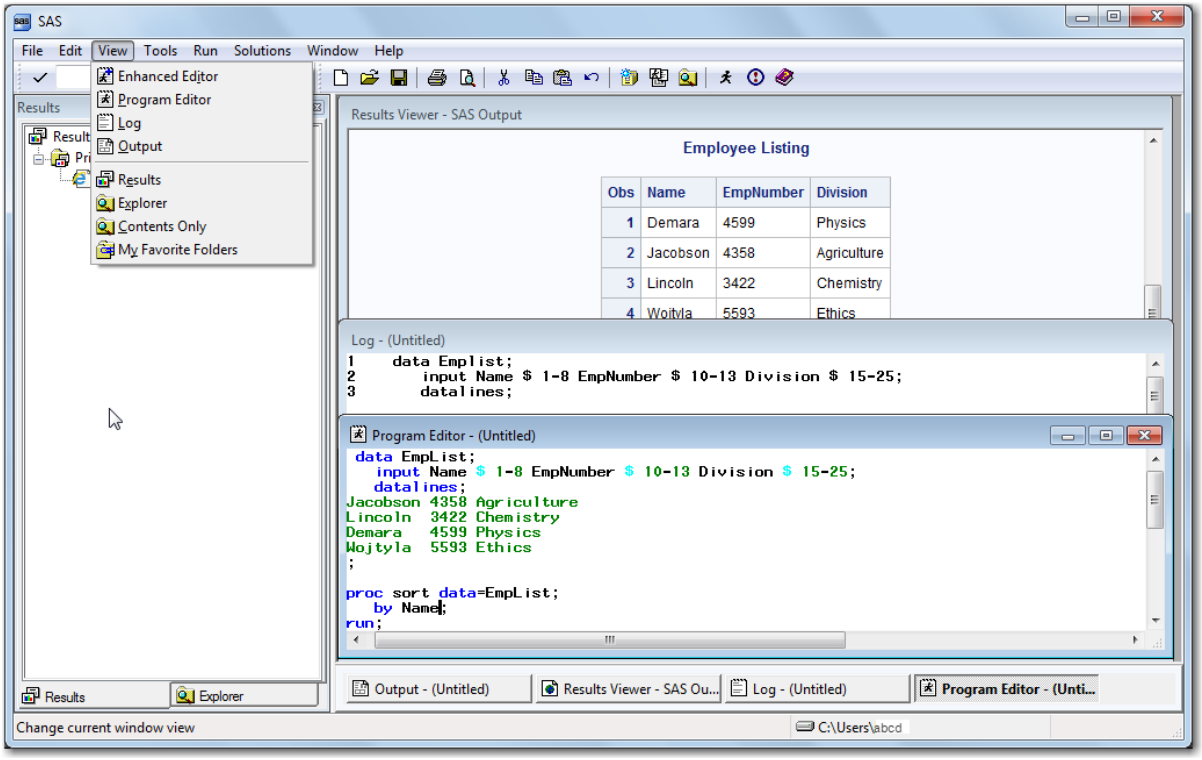
図 16.9 *エクスプローラウィンドウがアクティブな場合の表示オプション*



表示メニューから**プログラムエディタ**を選択し、再び**表示**を選択すると、**プログラムエディタ**ウィンドウがアクティブな場合に使用可能な**表示**オプションがメニューに表示されます。

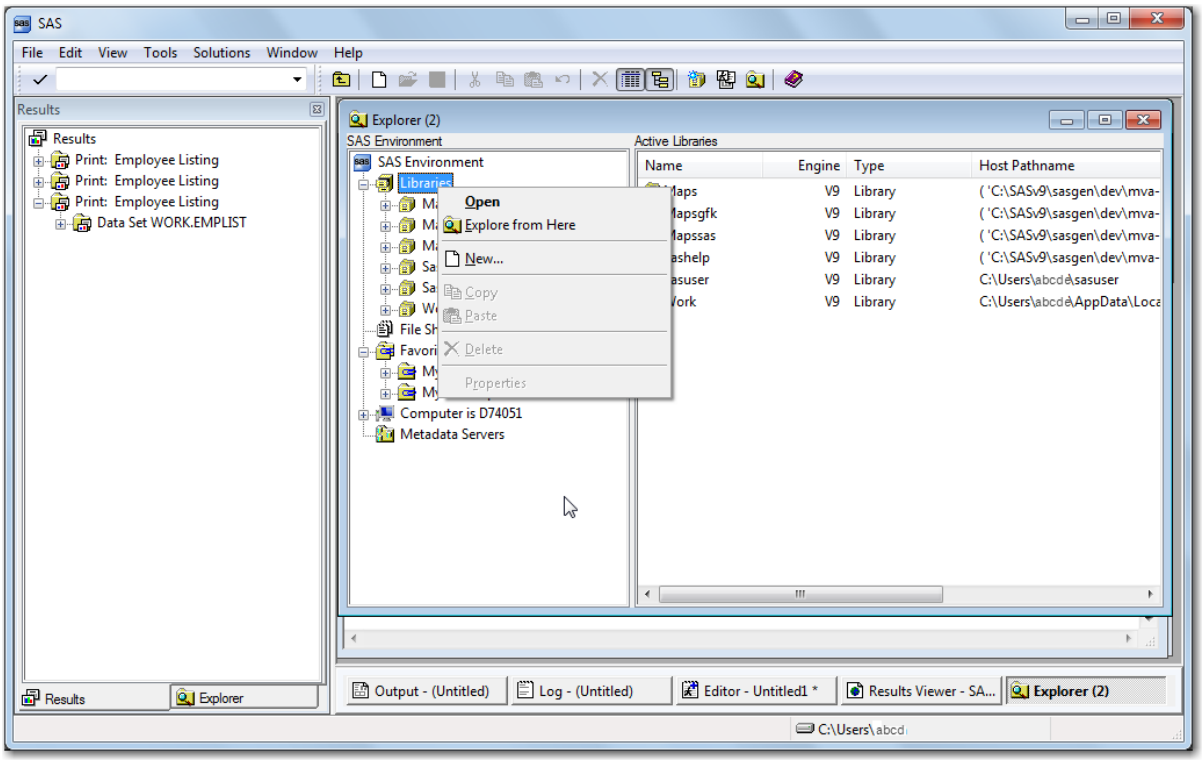
次のデ表示画面では、**プログラムエディタ**ウィンドウがアクティブになると、**表示**メニューが表示されます。

図 16.10 *プログラムエディタウィンドウがアクティブな場合の表示オプション*



項目を右クリックしてメニューにアクセスすることもできます。たとえば、**表示** ⇨ **エクスプローラ**を選択し、**エクスプローラ**ウィンドウで**ライブラリ**を右クリックすると、次のメニューが表示されます。

図 16.11 別のメニュー例



メニューは、メニュー項目を選択するか、メニュー領域外の領域をクリックするまで表示されます。

SAS のツールバー

ツールバーには、ウィンドウボタンまたはアイコンのブロックが表示されます。ツールバーで項目をクリックすると、機能またはアクションが開始されます。たとえば、ツールバーのプリンタの画像をクリックすると、印刷処理が始まります。ツールバーには、特定のウィンドウで頻繁に実行するさまざまなアクションのアイコンが表示されます。

z/OS 固有

z/OS 動作環境では、ツールバーは表示されません。詳細については、*z/OS 版 SAS* を参照してください。

表示されるツールバーは、どのウィンドウがアクティブになっているかによって異なります。たとえば、**プログラムエディタ**ウィンドウがアクティブな場合、次のツールバーが表示されます。

図 16.12 拡張エディタウィンドウがアクティブな場合の SAS ツールバーの例

ツールバーにある項目のいずれかにカーソルを置くと、アイコンの目的を示すテキストウィンドウが表示されます。

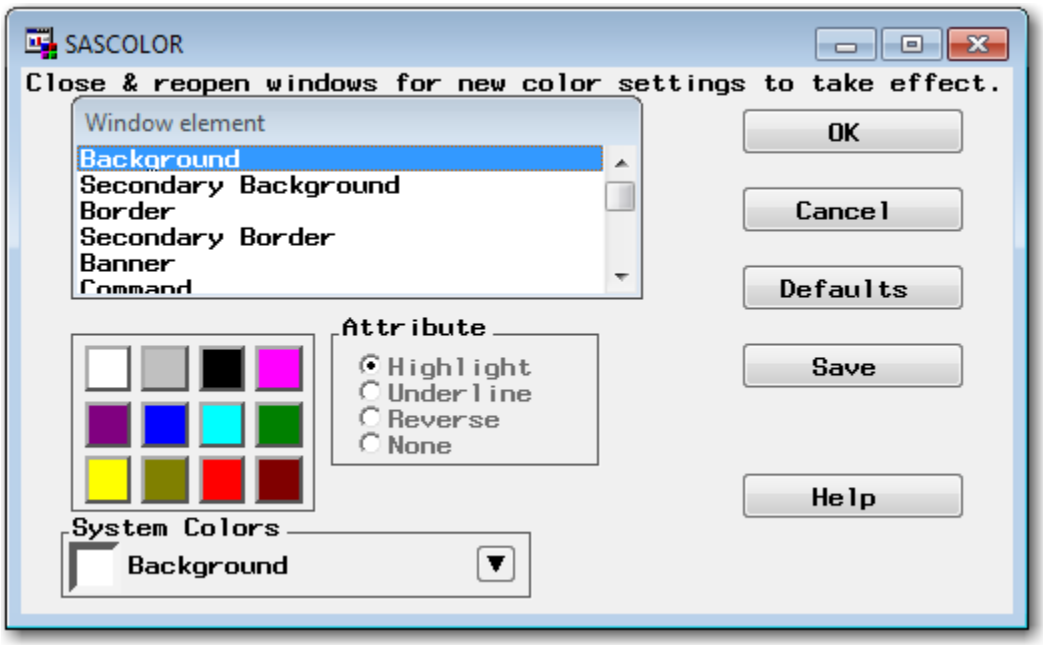
コマンドライン

コマンドラインは、ツールバーの左側にあります。コマンドラインでは、SAS ウィンドウを開いたり、ヘルプ情報を取得したりするコマンドが入力できます。

次に、**SASCOLOR** ウィンドウを開くコマンドの例を示します。

図 16.13 コマンドラインの例

図 16.14 SASCOLOR ウィンドウ

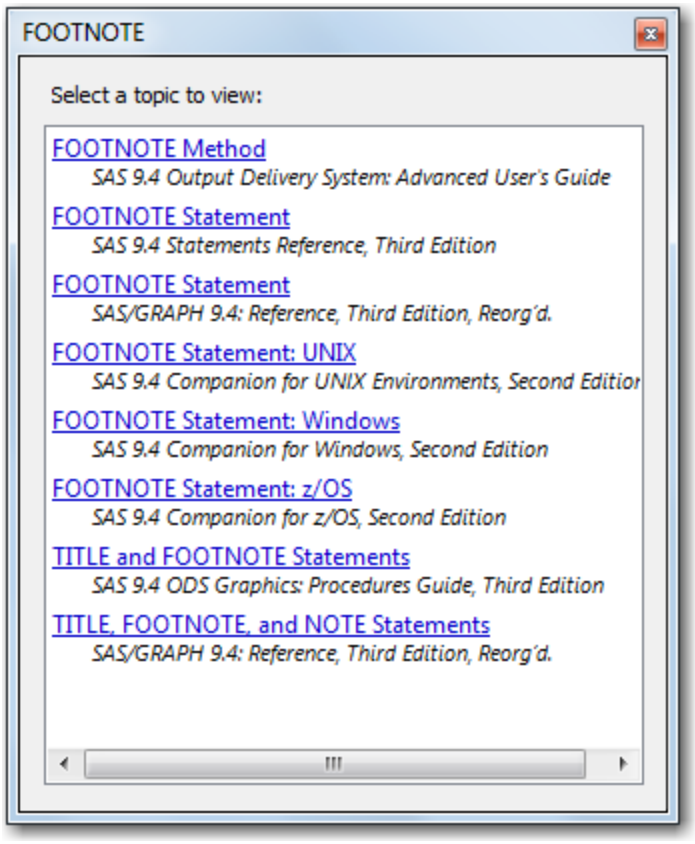


SAS でのヘルプの参照

コマンドラインで Help と入力

コマンドラインに Help と入力すると、アクティブウィンドウのヘルプが表示されます。`Help <item>` (たとえば、`Help footnote`)と入力すると、入力した項目のヘルプにアクセスできます。SAS セッションのコマンドラインに `Help footnote` と入力すると、次のウィンドウが表示されます。

図 16.15 SAS セッションのコマンドラインで Help と入力した結果



情報を含むドキュメントと一緒に、関連項目が表示されます。トピックをクリックするとその項目のヘルプが表示されます。

ツールバーからヘルプメニューを開く

ヘルプメニューを開くと、次の項目を選択できます。

このウィンドウの使用
: 現在のアクティブウィンドウについて説明するヘルプウィンドウを表示します。

SAS ヘルプとドキュメント
: SAS ヘルプとドキュメントシステムを開きます。システムにインストールした Base SAS およびその他の SAS 製品でヘルプを使用できます。情報を検索するには、目次の項目をクリックするか、項目を検索して結果をクリックします。

SAS ソフトウェア入門ガイド
: *SAS 入門ガイド*チュートリアルを表示します。このドキュメントを参照すると、SAS を使用する方法の基礎について学ぶことができます。

SAS プログラミングの学習
: オンライントレーニングライセンスを持っている場合、SAS オンライントレーニングを使用できます。このソフトウェアでは、初心者および経験のある SAS プログラマに対して、50 - 60 時間に及ぶ手順を表示します。

SAS Web サイト
: SAS Web サイトへのリンクを提供します。Web ページでは、次のことが可能です。

- テクニカルサポートに問い合わせることができます。
- トレーニングサービスに関する情報を検索できます。

- よくある質問(FAQ)を参照できます。
- フィードバックを送信できます。
- カスタマサポートセンタにアクセスできます。
- SAS Institute 社のホームページを参照できます。

SAS®9 について
SAS のバージョンとリリースの情報を参照できます。

個々の SAS ウィンドウでヘルプをクリックする

SAS ウィンドウを開いている場合、キーボードの HELP キー(通常は F1)を押すと、そのウィンドウに関する情報を表示できます。

SAS ウィンドウとウィンドウコマンドのリスト

基本的な SAS ウィンドウは、**エクスプローラ**、**結果**、**プログラムエディタ**、**拡張エディタ**(Windows 動作環境)、**ログ**、**出力**の各ウィンドウから構成されます。ただし、それ以外にも 30 以上のウィンドウがあり、SAS セッションの印刷や微調整などのタスクを実行できます。

次の表では、ウィンドウを開くポータルブル SAS ウィンドウ、ウィンドウの説明、コマンドを示します。

表 16.1 *SAS ウィンドウ、説明、ウィンドウコマンドのリスト*

ウィンドウ名	説明	ウィンドウコマンド
ドキュメント	階層ツリー構造で ODS ドキュメントを表示します。	ODSDOCUMENTS
配色の編集	編集ウィンドウのデフォルト色を変更できます。	SYNCONFIG
エクスプローラ	カタログ、ライブラリ、データセット、ホストファイルなどのデータへの一括アクセスポイントを提供します。	ACCESS、BUILD、CATALOG、DIR、EXPLORER、FILENAME、LIBNAME、V6CAT、V6DIR、V6FILENAME、V6LIBNAME
エクスプローラオプション	ファイルタイプを追加または削除し、ポップアップメニュー項目を変更または追加し、エクスプローラに表示されるフォルダを選択し、メンバの詳細を表示します。	DMEXPOPTS
ファイルショートカットの作成	グラフィカルユーザーインターフェイスを使用してファイルショートカットをファイルに割り当てます。	DMFILEASSIGN

ウィンドウ名	説明	ウィンドウコマンド
検索	SAS ライブラリの式を検索できます。	EXPFIND
フォント選択 (動作環境に固有)	フォント、フォントスタイル、フォントサイズを選択できます。	DLGFONT
FOOTNOTES	出力のフットノートを入力、参照、変更できます。	FOOTNOTES
FSBROWSE	参照用のデータセットを選択できます。	FSBROWSE
FSEDIT	FSEDIT プロシジャによって処理するデータセットを選択できます。	FSEDIT
FSFORM	出力をプリンタに送信するためのフォームをカスタマイズできます。	FSFORM *formname*
FSLETTER	カタログエントリを編集または作成できます。	FSLETTER
FSLIST	SAS セッションで外部ファイルを参照できます。	FSLIST
FSVIEW	行と列を備えた表としてデータセットを表示し、SAS データセットを参照、編集、作成できます。	FSVIEW
HELP	SAS に関するヘルプ情報を表示します。	HELP
KEYS	ファンクションキー設定を参照、変更、保存できます。	KEYS
ログ	現在の SAS セッションのメッセージと SAS ステートメントを表示します。	LOG
Metabase	SAS/EIS メタベース機能にアクセスし、データ登録のコピー、リポジトリファイルの作成、削除、編集を実行します。	METABASE
メタデータブラウザ	**メタデータサーバーの構成**ダイアログボックスを表示します。	METABROWSE (z/OS では無効)

ウィンドウ名	説明	ウィンドウコマンド
Metafind	URI (Uniform Resource Identifier)を使用してリポジトリのメタデータオブジェクトを検索できます。	METAFIND
メタデータサーバーの接続	メタデータ接続をインポート、エクスポート、追加、削除、並べ替え、テストできます。	METACON
ライブラリ作成	新規 SAS ライブラリを作成し、ライブラリ参照名を割り当てることができます。	DMLIBASSIGN
NOTEPAD	テキストの Notepad を作成、格納できます。	NOTEPAD、NOTE、FILEPAD *filename*
オプション(SAS システムオプション)	一部の SAS システムオプションを表示および変更できます。	OPTIONS
出力	リスト形式でプロシジャ出力を表示します。	OUTPUT、OUT、LISTING、LIST、LST
ページ設定	ユニバーサル印刷ジョブに適用するページ設定オプションを指定できます。	DMPAGESETUP
パスワード	特定のデータセットのパスワードを編集、割り当て、消去できます。	SETPASSWORD (に続いて 2 レベルのデータセット名)
プリファレンス (動作環境に固有)	SAS System プリファレンスを設定または編集できます。	DLGPREF
印刷	ユニバーサル印刷を使用してアクティブな SAS ウィンドウの内容を印刷できます。	DMPRINT
印刷設定	デフォルトプリンタの変更、プリンタ定義の作成または編集、ユニバーサル印刷のプリンタ定義の削除が可能です。	DMPRTSETUP
プログラムエディタ	SAS ステートメントを入力、編集、サブミットし、ソースファイルを保存できます。	PROGRAM、PGM
プロパティ	現在のデータセットに関連付けられている詳細を表示します。	VAR *libref.SAS-data-set*, V6VAR *libref.SAS-data-set*

ウィンドウ名	説明	ウィンドウコマンド
SAS レジストリエディタ	SAS レジストリを編集し、SAS ウィンドウ環境のさまざまな側面をカスタマイズできます。	REGEDIT
結果	SAS によって生成されるプロシジャ出力をリストします。	ODSRESULTS
SAS/ACCESS		ACCESS
SAS/AF	SAS/AF ソフトウェアで作成されたウィンドウアプリケーションを表示します。	AF、AFA
SAS/ASSIST	SAS の使用を単純化する、SAS/ASSIST ソフトウェアのプライマリメニューを表示します。	ASSIST
SASCOLOR	SAS ウィンドウの異なるウィンドウ要素のデフォルト色を変更できます。	SASCOLOR
SQL QUERY	SQL (Structured Query Language)に習熟していなくとも、クエリを構築、実行、保存できます。	QUERY
SAS システムオプション	SAS システムオプション設定を変更できます。	OPTIONS
テンプレート	テンプレートソースコードを参照、編集できます。	ODSTEMPLATES
TITLES	出力のタイトルを入力、参照、変更できます。	TITLES
VIEWTABLE	表(データセット)の参照、編集、作成を行うことができます。	VIEWTABLE、VT

注: 動作環境に固有の追加 SAS ウィンドウも使用可能になる可能性があります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

17 章

SAS ウィンドウ環境を用いたデータ管理

SAS ウィンドウ環境におけるデータ管理の概要

SAS ウィンドウ環境には、コードを記述しなくても一般的なデータ操作や変更が可能なウィンドウがあります。

SAS に習熟していない場合、または SAS 言語によるコード記述に習熟していない場合は、ウィンドウ環境が便利でしょう。ウィンドウ環境では、データセットを開き、データの行と列をポイントできます。その後、メニュー項目をクリックして、その情報を再編成したり分析を実行したりできます。

SAS ウィンドウ環境の詳細については、SAS セッションを呼び出した後で**ヘルプ**メニューから **SAS ヘルプとドキュメント**を選択します。

SAS エクスプローラを用いたデータ管理

SAS エクスプローラを用いたデータ管理について

SAS **エクスプローラ**を使用して、データセットの表示や管理が行えます。データセットは、SAS ファイルとカタログの保管場所であるライブラリに格納されます。デフォルトで、SAS は次のような複数のライブラリを定義します。

Sashelp
: ヘルプウィンドウ、デフォルトファンクションキー定義、ウィンドウ定義、メニューのテキストを格納する、SAS で作成されるライブラリです。

Maps
: 地理またはその他の領域のグラフィカル表現を表す SAS で作成されるライブラリです。

Sasuser
: 最初の SAS セッションの最初に作成される永久 SAS ライブラリです。このライブラリは、SAS 用に指定したカスタマイズ済み機能または設定を保存するプロファイルカタログを格納します。(別の SAS ファイルをこのライブラリに保存できます)。

Work
: 各 SAS セッションまたは SAS ジョブの最初に SAS によって作成されるライブラリです。ユーザーライブラリを指定しなかった場合、デフォルトでは、1 レベルの名前が付いた SAS ファイルが新しく作成されて Work ライブラリに置かれます。新しく作成されたファイルは、現在のセッションまたはジョブの終了時に削除されます。

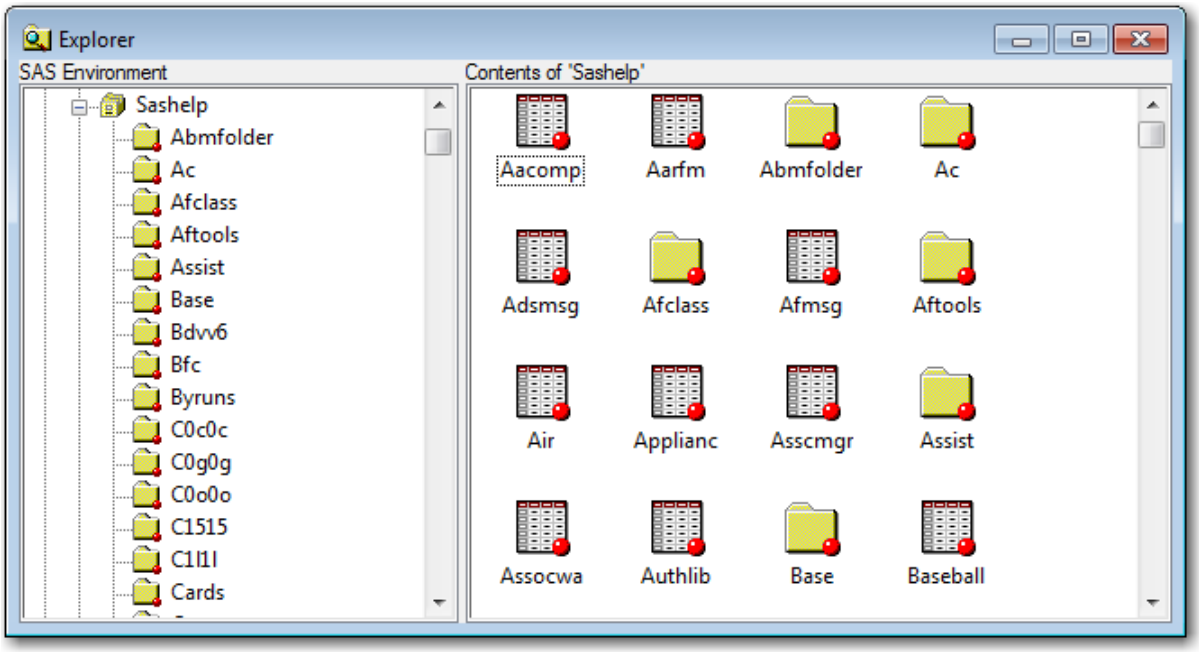
ライブラリとデータセットの表示

ライブラリとデータセットは、大きなアイコン、小さなアイコン、一覧のいずれかで表されます。**エクスプローラ**ウィンドウをアクティブにし、**表示**メニューからオプションを選択して、表示を変更することができます。

- 大きなアイコンを表示するには、**表示**メニューから**大きいアイコン**を選択します。
- 小さいアイコンを表示するには、**表示**メニューから**小さいアイコン**を選択します。
- データセットを一覧で表示するには、**表示**メニューから**一覧**を選択します。

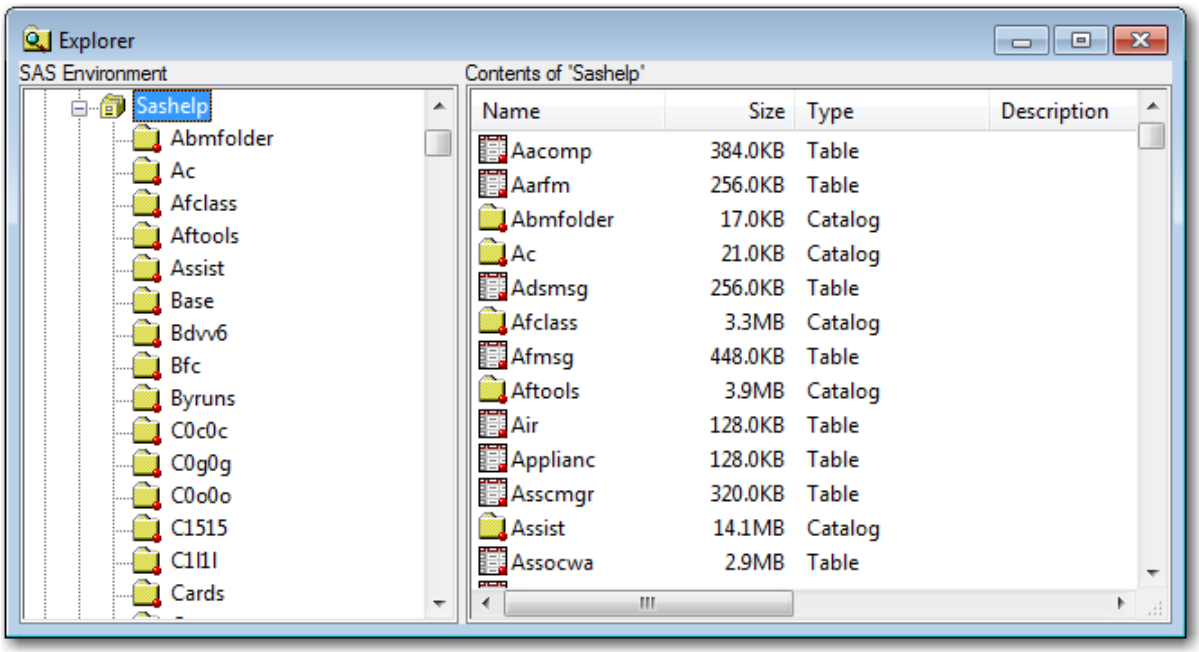
次の例では、大きいアイコンで Sashelp の内容を表示しています。

図 17.1 大きいアイコンで表された Sashelp ライブラリ



Sashelp ライブラリを選択して、メニューバーから**表示** ⇨ **詳細**を選択すると、Sashelp ライブラリの内容がデータセットのサイズおよび種類と一緒に表示されます。

図 17.2 Sashelp ライブラリの詳細表示



このリストのテーブルをダブルクリックすると、データセットが表示されます。SAS テーブルビューアおよびエディタである **VIEWTABLE** ウィンドウが表示され、テーブルのデータが挿入されます。

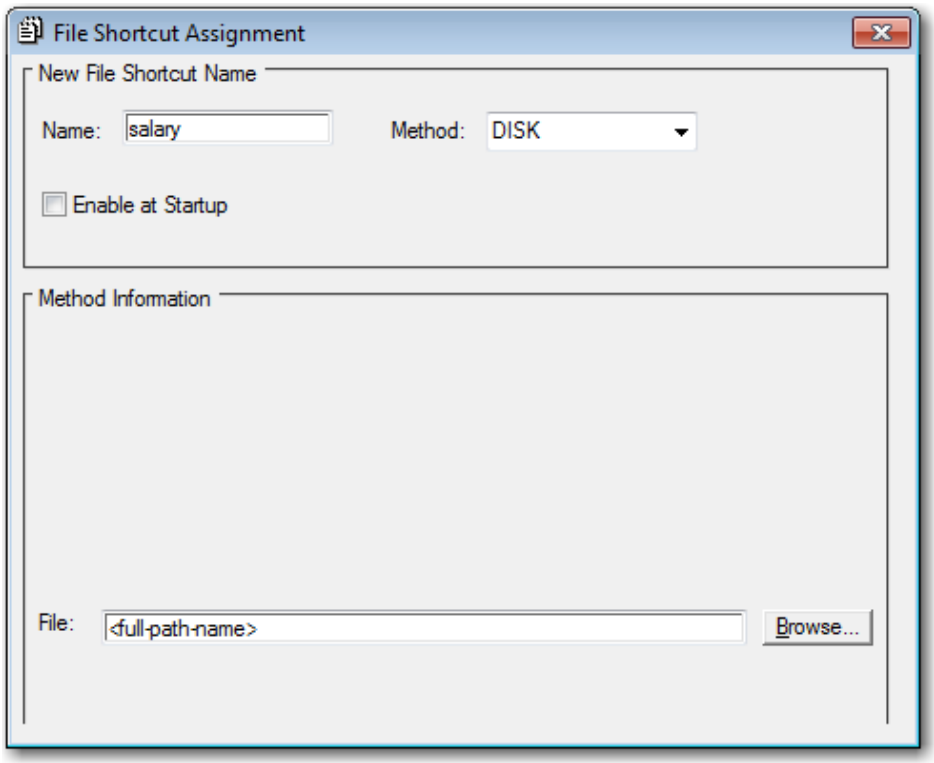
ファイルショートカットの割り当て

ファイルショートカットは、ファイル参照名(fileref)とも呼ばれます。ファイル参照名を使用すると、よく使用するファイルにニックネームを割り当てることで、プログラムにかかる時間を減らせます。FILENAME ステートメントを使用すると、ファイル参照名を作成できます。または、SAS **エクスプローラ**の**ファイルショートカットの作成**ウィンドウを使用できます。

ファイル参照名をファイルに割り当てるには、次の操作を実行します。

1. メニューから**ツール** ⇨ **ファイルショートカットの作成**を選択します。
2. 表示された**ファイルショートカットの作成**ウィンドウで、使用するファイル参照名を**名前**フィールドに入力します。
3. ファイルの絶対パス名を**ファイル**フィールドに入力します。

   次に、**ファイルショートカットの作成**ウィンドウが表示されています。



デフォルトでは、作成したファイル参照名は一時的なもので、現在の SAS セッションでのみ使用できます。ただし、**起動時に有効**を**ファイルショートカットの割り当て**ウィンドウから選択すると、新しい SAS セッションを開始するたびにファイル参照名がファイルに割り当てられます。

SAS データセットの名前変更

書き込み保護されていない場合、SAS ライブラリのデータセット名を変更できます。データセットを名前変更するには、次の操作を実行します。

1. SAS **エクスプローラ**を起動し、ライブラリを選択します。

   ライブラリの内容が右ペインに表示されます。

2. 名前変更するデータセットを右クリックします。
3. メニューから**名前の変更**を選択し、データセットの新しい名前を入力します。
4. **OK** をクリックします。

SAS データセットのコピーまたは複製

SAS データセットを別のライブラリまたはカタログにコピーできます。または、元のデータセットと同じディレクトリにデータセットの複製を作成できます。データセットをコピーまたは複製するには、次の操作を実行します。

1. SAS **エクスプローラ**を起動し、ライブラリを選択します。

   ライブラリの内容が右ペインに表示されます。
2. コピーまたは複製するデータセットを右クリックします。
3. 表示されるメニューから**コピー**を選択してデータセットを別のライブラリまたはカタログにコピーするか、**複製**を選択してデータセットを同じライブラリまたはカタログにコピーします。
4. **コピー**を選択する場合は、次の操作を行います。
   a. SAS **エクスプローラ**の左ペインでライブラリをクリックし、データセットのコピー先のライブラリまたはカタログを選択します。
   b. 右ペインで、マウスを右クリックし、表示されるメニューから**貼り付け**を選択します。

      データセットのコピーは、新しいディレクトリに格納されます。
5. **複製**を選択すると、**複製**ウィンドウが表示されます。**複製**ウィンドウで、データセット名に**_copy** が追加されます(たとえば、**data-set-name_copy** のようになります)。

   次のいずれかの操作を実行します。
   - 名前を保持し、**OK** をクリックします。
   - 複製されたデータセットに別の名前を付け、**OK** をクリックします。

ライブラリ内のデータセットの並べ替え

SAS **エクスプローラ**のデータセットは、自動的に名前順に並べ替えられます。データセットの並べ替え順序をサイズ別または種類別に変更するには、**サイズ**列または**種類**列をクリックします。データセットの並べ替え順序を元に戻すには、**表示**メニューから**最新の情報に更新**オプションを選択します。

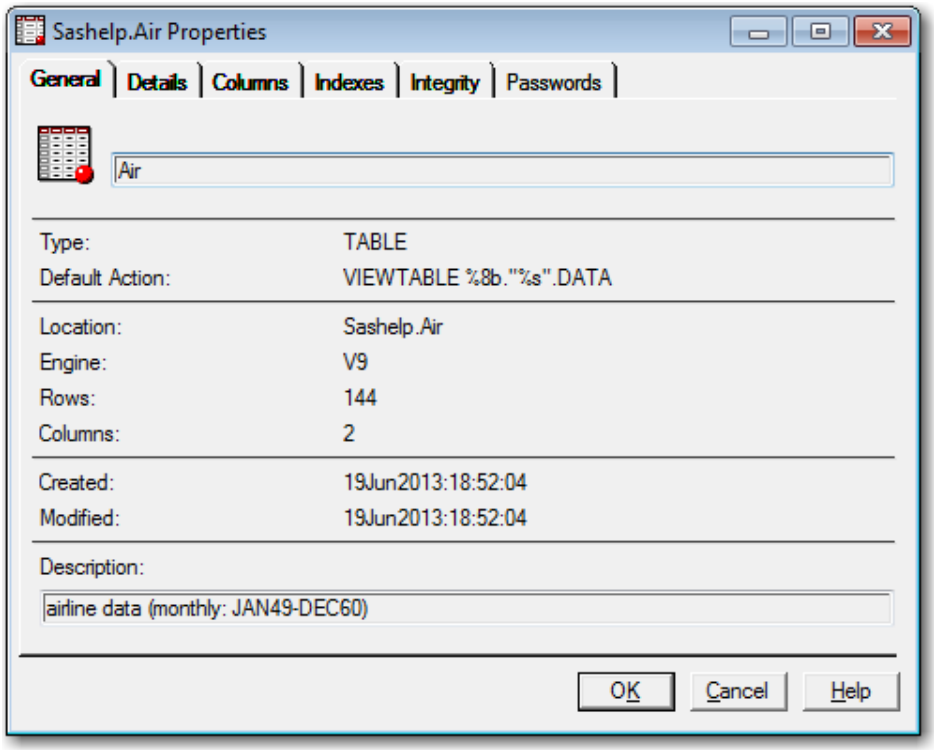
SAS データセットのプロパティの表示

データセットのプロパティを表示するには、**プロパティ**ウィンドウを使用します。プロパティを表示するには、次の操作を実行します。

1. SAS **エクスプローラ**を起動し、ライブラリを選択します。

   ライブラリの内容が右ペインに表示されます。
2. 表示するデータセットを右クリックします。
3. メニューから**プロパティ**を選択します。

データセット Air の場合、次のようなウィンドウが表示されます。



4. **全般**タブの**説明**フィールドで、データセットの説明を入力できます。説明を保存するには、**OK** をクリックします。
5. データセットに関する追加情報を表示するには、他のタブを選択します。

VIEWTABLE の操作

VIEWTABLE の概要

データを対話的に操作するには、SAS テーブルエディタの **VIEWTABLE** を使用します。**VIEWTABLE** ウィンドウでは、新しいテーブルの作成、および既存のテーブルの表示または編集が可能です。

この短いビデオでは、SAS エクスプローラを使用して VIEWTABLE を開く方法を説明します。

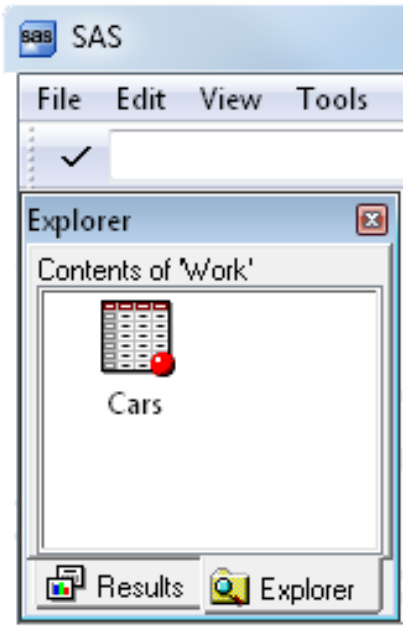
VIEWTABLE ウィンドウで SAS データセットを開く

VIEWTABLE ウィンドウで既存の SAS データセットを開くには、SAS エクスプローラウィンドウで SAS データセットアイコンをダブルクリックするか、SAS ディスプレイマネージャのコマンドラインで VIEWTABLE コマンドを指定します。次のセクションでは、この 2 つの方法の使用について詳細を説明します。

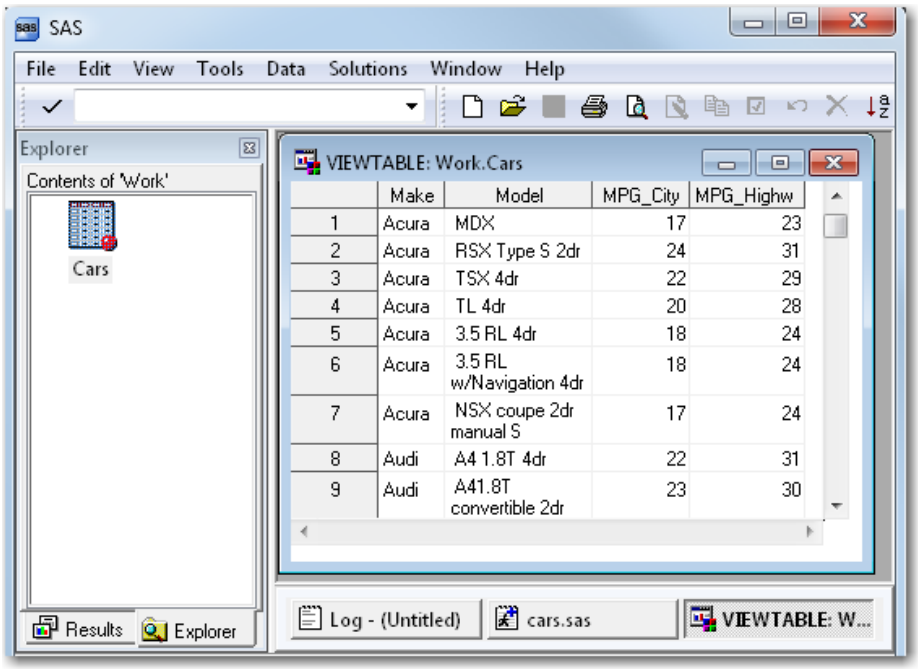
SAS エクスプローラウィンドウの使用

SAS エクスプローラウィンドウを使用して **VIEWTABLE** ウィンドウで SAS データセットを開くためのステップを次に示します。

1. SAS **エクスプローラ**を開いて、対象データセットを含むライブラリのアイコンをダブルクリックします。
2. 任意のデータセットを選択し、そのアイコンをダブルクリックします。



3. **VIEWTABLE** ウィンドウが表示され、データセットからのデータが格納されます。



4. **VIEWTABLE** ウィンドウでスクロールバーを使用して、すべてのデータを表示します。

VIEWTABLE コマンドの使用

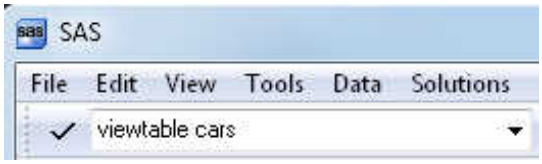
SAS ディスプレイマネージャのコマンドラインで VIEWTABLE コマンドを使用して、VIEWTABLE ウィンドウでデータセットを開くこともできます。

1. SAS ディスプレイマネージャのコマンドラインで次の構文を使用して VIEWTABLE コマンドを指定します。

   VIEWTABLE *data-set-name <-options>*

2. 次に例を示します。

```
viewtable cars
```

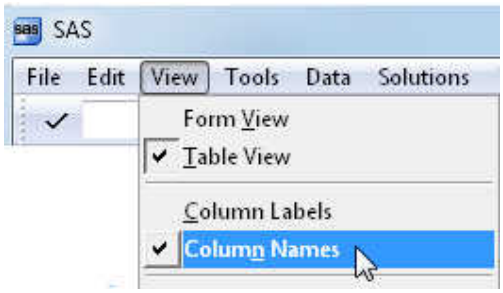


テーブルヘッダーを名前またはラベルとして表示

SAS では、VIEWTABLE ウィンドウでラベルを含むデータセットを開くと、自動的にテーブルヘッダーが変数名ではなく変数ラベルとして表示されます。SAS のテーブルヘッダーの表示方法を変更するには、VIEWTABLE ポップアップメニューを使用するか、または VIEWTABLE コマンドを使用します。

- VIEWTABLE ポップアップメニューを使用してテーブルヘッダーの表示方法を変更するには、次の操作を行います。

  1. VIEWTABLE でデータセットを開きます(VIEWTABLE ポップアップメニューにアクセスするには、アクティブな VIEWTABLE ウィンドウが開いている必要があります)。
  2. VIEWTABLE ウィンドウがアクティブであることを確認します。
  3. **表示** ⇨ **列名**または**表示** ⇨ **列ラベル**をドロップダウン**表示**メニューから選択します。



  4. ここで選択を行うと、開いているテーブルとこの後で開くすべてのテーブルにおいて、VIEWTABLE ポップアップメニューのこの設定に基づいてテーブルヘッダーが表示されます。VIEWTABLE または SAS を終了する際に、列ラベルや列名に関するプリファレンスが保存されます。その後、VIEWTABLE を再度開くか、または SAS を再度起動すると、事前に選択したプリファレンスが自動的に選択されます。

     この機能は、SAS 9.4M1 およびそれ以降のバージョンで利用できます。

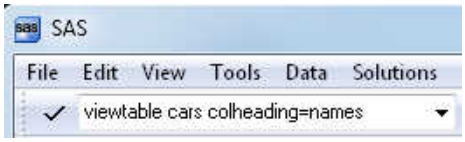
- VIEWTABLE コマンドを使用して、テーブルを開く際のテーブルヘッダーの表示方法を変更するには、次の操作を行います。

  1. SAS コマンドラインで次の構文を使用して、VIEWTABLE コマンドで COLHEADING=オプションを指定します。

  VIEWTABLE *data-set-name* -<COLHEADING=NAMES | LABELS>

2. 次に例を示します。

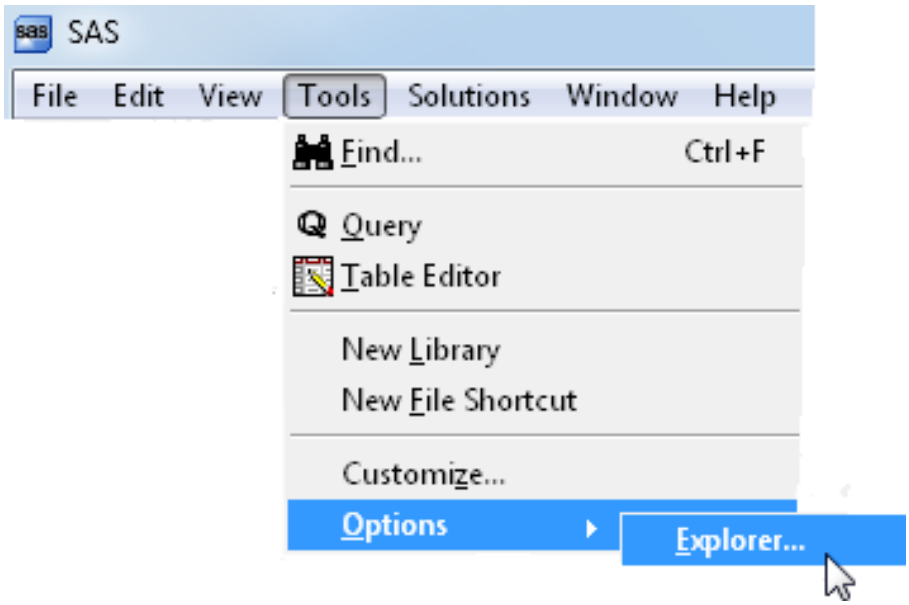
```
viewtable cars colheading=names
```



VIEWTABLE ウィンドウを開くための SAS エクスプローラのカスタマイズ

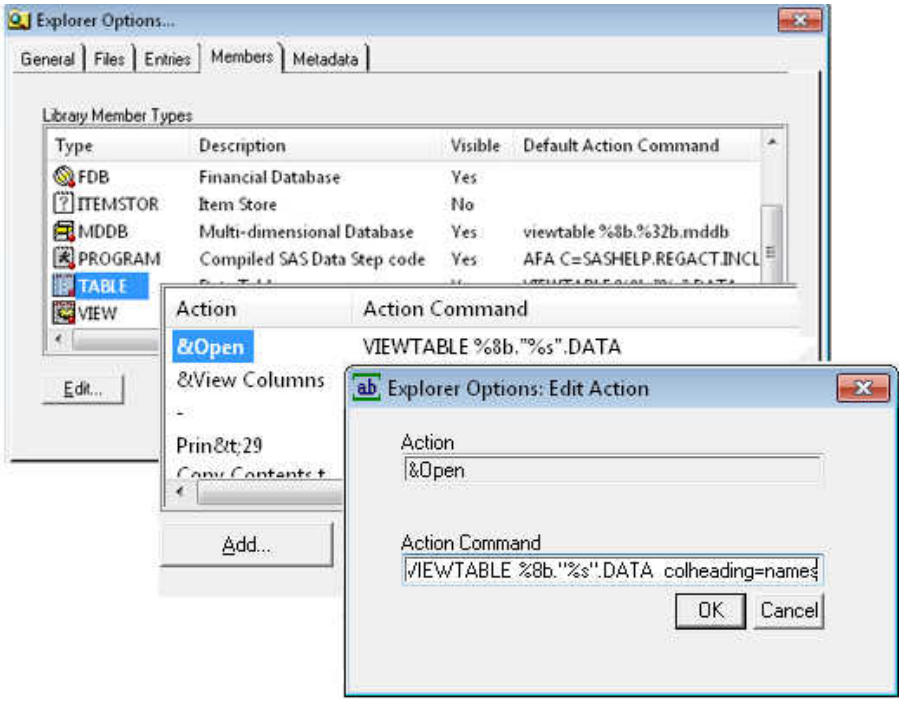
SAS エクスプローラをカスタマイズして VIEWTABLE ウィンドウを開くようにできるので、これにより SAS エクスプローラウィンドウからテーブルを開くたびに列見出しを名前かラベルのどちらかとして表示できます。これを行うには、**SAS エクスプローラ オプション**ダイアログボックスで**アクションコマンド**に COLHEADING=オプションを追加します。

1. SAS エクスプローラウィンドウをアクティブにし、**ツール** ⇨ **オプション** ⇨ **エクスプローラ**を選択して、エクスプローラオプションウィンドウを開きます。



2. **メンバ**タブを選択します。
3. 登録タイプのリストで**テーブル**を選択して、**編集**をクリックすると、**TABLE オプションダイアログボックス**が開きます。
4. アクションのリストで**開く(O)**アクションコマンドを選択してから**編集**をクリックすると、**アクションの編集**ダイアログボックスが開きます。
5. **アクションの編集**ダイアログボックスで、VIEWTABLE コマンドの最後に**-COLHEADING=<*value*>**を追加します。

```
VIEWTABLE %8b.'%s'.DATA colheading=names
```



6. 選択が終了したら、OK を 3 回クリックして、開いているダイアログボックスをすべて終了します。この時点から、SAS エクスプローラウィンドウを使用して VIEWTABLE ウィンドウを開くと、この SAS エクスプローラダイアログボックスでの指定に従ってテーブルヘッダーが表示されるようになります。

注: これらのステップは、SAS **エクスプローラ**ウィンドウから開く(アイコンをダブルクリックするか、アイコンを右クリックして"開く"を選択する)場合のテーブルの表示にのみ影響します。VIEWTABLE コマンドを使用してテーブルを開く場合のテーブルの開き方には影響しません。

列見出しの表示の優先順位

VIEWTABLE コマンドを使用してテーブルを開き、なおかつ列見出しの表示を制御するための COLHEADING=を指定しない場合、SAS では、VIEWTABLE ポップアップメニュー(**表示** ⇨ **列名**または**表示** ⇨ **列ラベル**)で前回設定した内容に基づいて列見出しが表示されます。

VIEWTABLE colheading=<*value*>コマンドを使用してテーブルを開く場合は、SAS では、VIEWTABLE ポップアップメニューでの列見出しの設定内容に関係なく、COLHEADING 値に従って列見出しが表示されます。VIEWTABLE ポップアップメニューの設定には、COLHEADING=値が反映されます。言い換えれば、COLHEADING=が、VIEWTABLE ポップアップメニューで指定された設定よりも優先されます。

SAS の LABEL ステートメントの詳細については、“LABEL Statement” (*SAS Statements: Reference*)を参照してください。

VIEWTABLE コマンドのファンクションキーへのマッピング

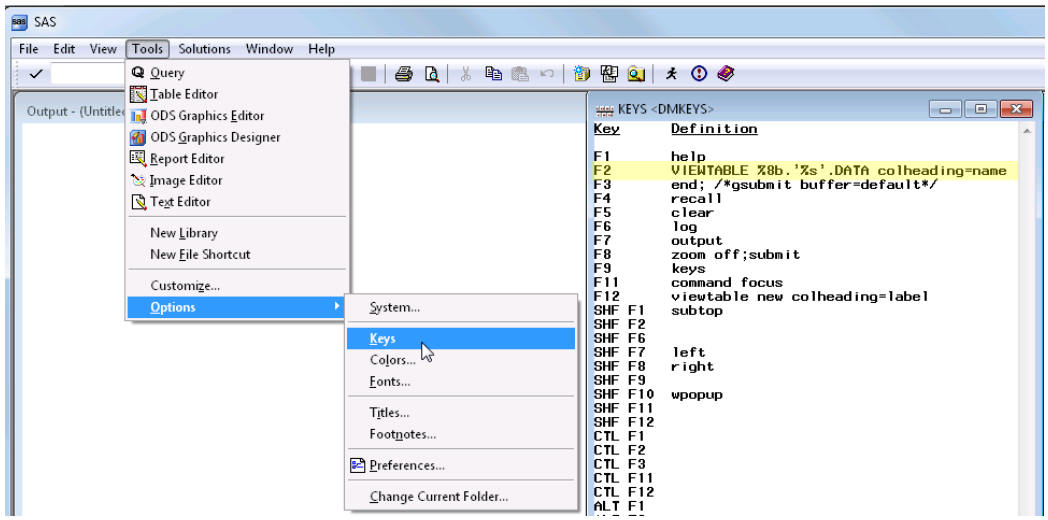
ディスプレイマネージャキーウィンドウでファンクションキーをマップして、VIEWTABLE コマンドを実行することができます。これには、次の操作を実行します。

1. SAS メニューから**ツール** ⇨ **オプション** ⇨ **キー**を選択します。キーウィンドウが表示されます。

2. キーウィンドウで、VIEWTABLE コマンドに割り当てる F キーを選択し、選択した F キーの定義フィールドにカーソルを合わせます。
3. VIEWTABLE コマンドと任意のオプションを入力します。次に例を示します。

```
VIEWTABLE %8b. '%s'.DATA colheading=name
```

4. キーウィンドウを閉じます。



VIEWTABLE の使用方法の詳細については、次のドキュメントを参照してください。

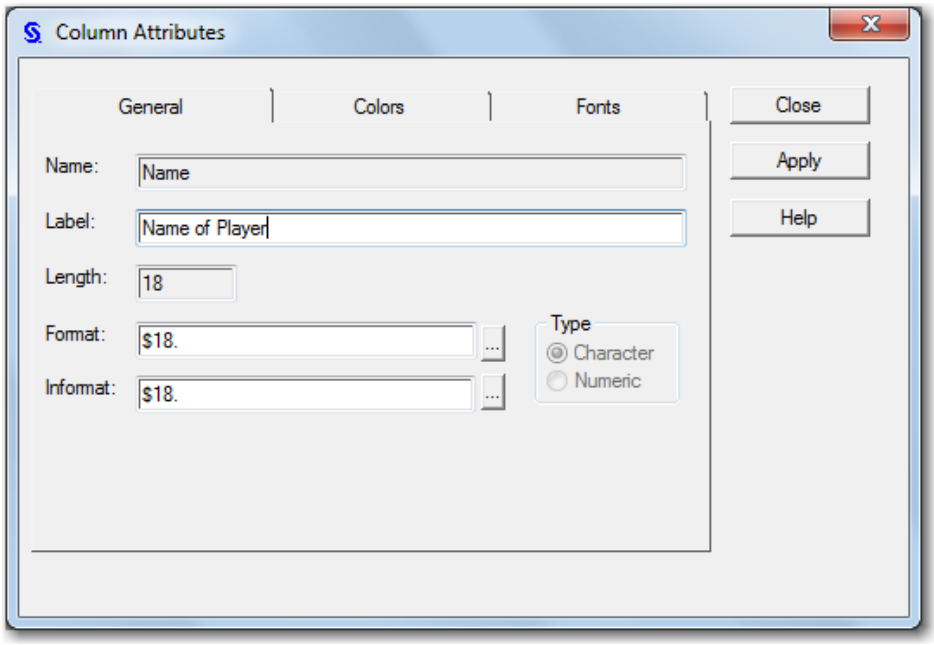
- Doing More with the SAS® Display Manager:From Editor to ViewTable - Options and Tools You Should Know (PDF)
- Sample 25263:Customizing open actions for the VIEWTABLE window for SAS® Explorer (HTML)

列見出しの一時変更

VIEWTABLE ウィンドウ内で、列見出しを一時的に変更できます。列見出しを一時的に変更するには、次の操作を実行します。

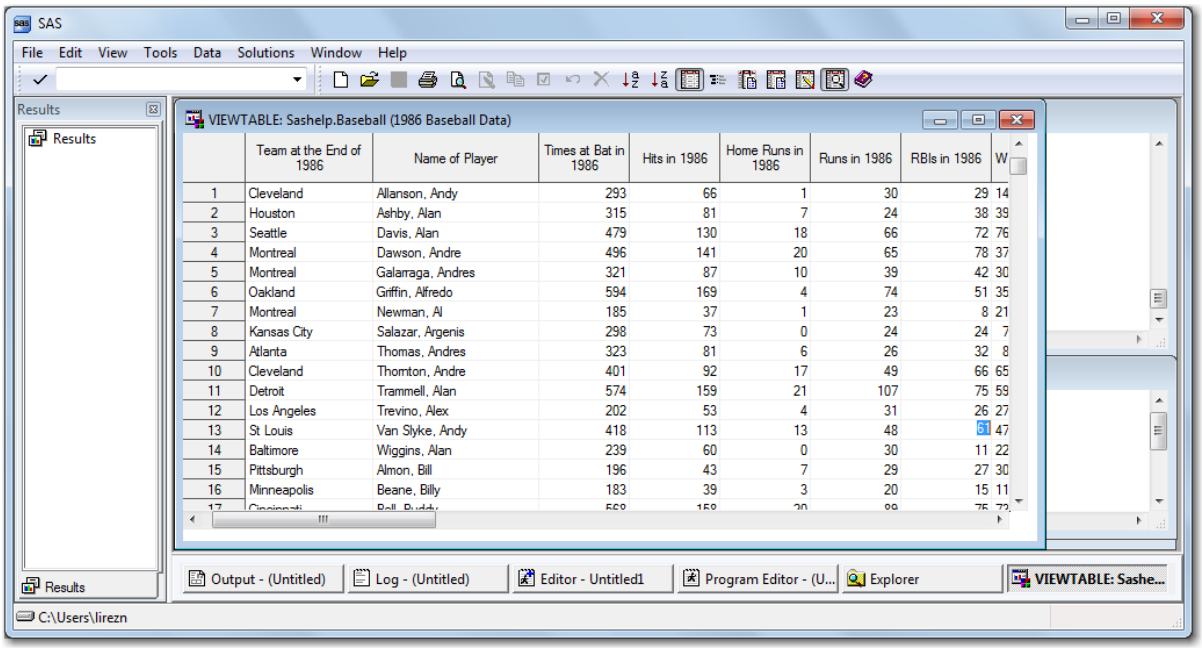
1. 変更する列の見出しを右クリックし、メニューから**列属性**を選択します。
2. **列属性**ウィンドウの**ラベル**フィールドに、列見出しの新しい名前を入力し、**適用**をクリックします。

   次の例では、見出し Name を、Name of Player というラベルで置き換えます。



適用を押すと、**VIEWTABLE** の列見出しが新しい名前に変更されます。

次の画面では、ラベルが **Name of Player** に変更されています。



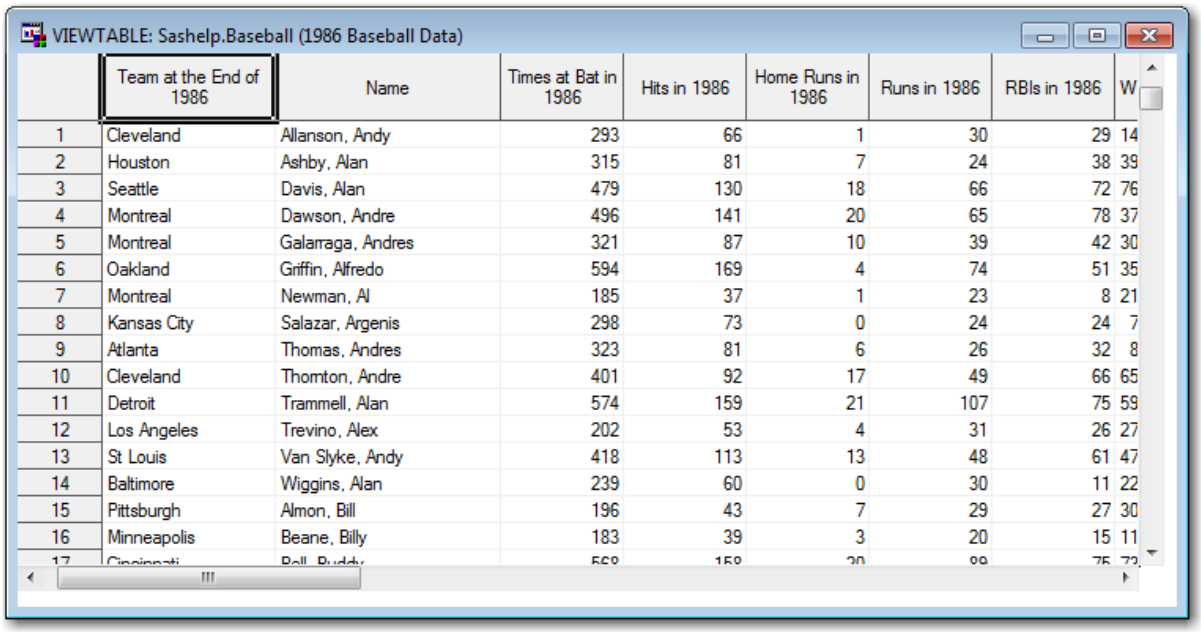
3. **閉じる**をクリックし、**列属性**ウィンドウを閉じます。

テーブルの列の移動

VIEWTABLE ウィンドウ内で、テーブル内の列を再配置できます。テーブルの列を移動するには、次の操作を実行します。

1. 移動する列見出しをクリックします。
2. 見出しを別の列見出しにドラッグアンドドロップします。

次の例で、Name という見出しをクリックし、Name を Team at the End of 1986 にドラッグアンドドロップした場合、Name 列が Team at the End of 1986 列の右側に移動します。



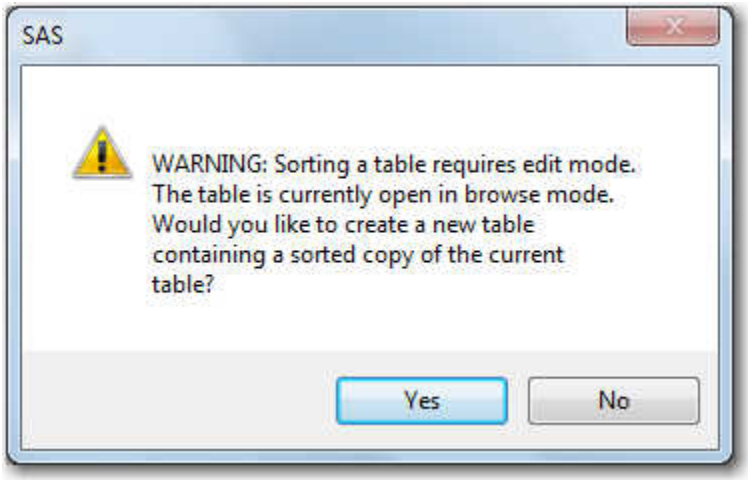
	Team at the End of 1986	Name	Times at Bat in 1986	Hits in 1986	Home Runs in 1986	Runs in 1986	RBIs in 1986	W
1	Cleveland	Allanson, Andy	293	66	1	30	29	14
2	Houston	Ashby, Alan	315	81	7	24	38	39
3	Seattle	Davis, Alan	479	130	18	66	72	76
4	Montreal	Dawson, Andre	496	141	20	65	78	37
5	Montreal	Galarraga, Andres	321	87	10	39	42	30
6	Oakland	Griffin, Alfredo	594	169	4	74	51	35
7	Montreal	Newman, Al	185	37	1	23	8	21
8	Kansas City	Salazar, Argenis	298	73	0	24	24	7
9	Atlanta	Thomas, Andres	323	81	6	26	32	8
10	Cleveland	Thornton, Andre	401	92	17	49	66	65
11	Detroit	Trammell, Alan	574	159	21	107	75	59
12	Los Angeles	Trevino, Alex	202	53	4	31	26	27
13	St Louis	Van Slyke, Andy	418	113	13	48	61	47
14	Baltimore	Wiggins, Alan	239	60	0	30	11	22
15	Pittsburgh	Almon, Bill	196	43	7	29	27	30
16	Minneapolis	Beane, Billy	183	39	3	20	15	11

列の値を基準に並べ替える

列の値に基づいて、昇順または降順でテーブルを並べ替えることができます。データを恒久的に並べ替えたり、テーブルの並べ替え後のコピーを作成したりすることが可能です。

テーブルを並べ替えるには、次の操作を実行します。

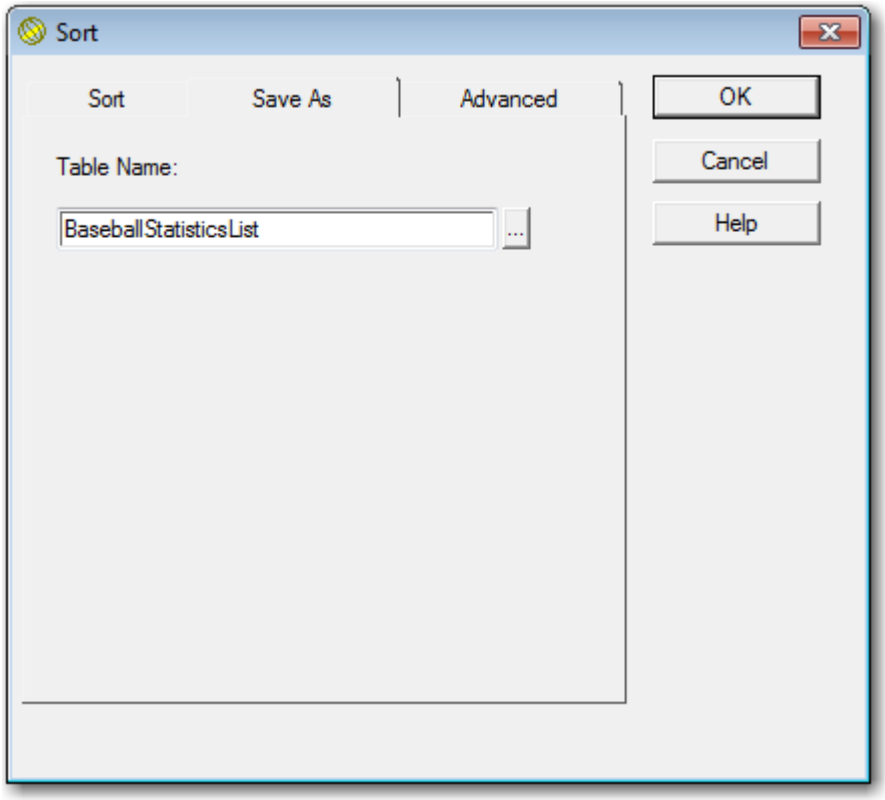
1. 並べ替える列の見出しを右クリックし、メニューから**並べ替え**を選択します。
2. メニューから**昇順**または**降順**を選択します。
3. 次の警告メッセージが表示されたら、**はい**をクリックし、並べ替え後のテーブルのコピーを作成します。



注: テーブルを開いてデータセルをクリックしてから**編集モード**を選択した場合、このウィンドウは表示されません。元のテーブルが更新されます。

4. **並べ替え**ウィンドウで、並べ替えた新しいテーブルの名前を入力します。

   次の例では、並べ替えたテーブルの名前は **BaseballStatisticsList** です。



5. **OK** をクリックします。

   新しいテーブルの行が、**Team at the End of 1986** の値で昇順に並べ替えられます。

VIEWTABLE: BaseballStatisticsList (1986 Baseball Data)

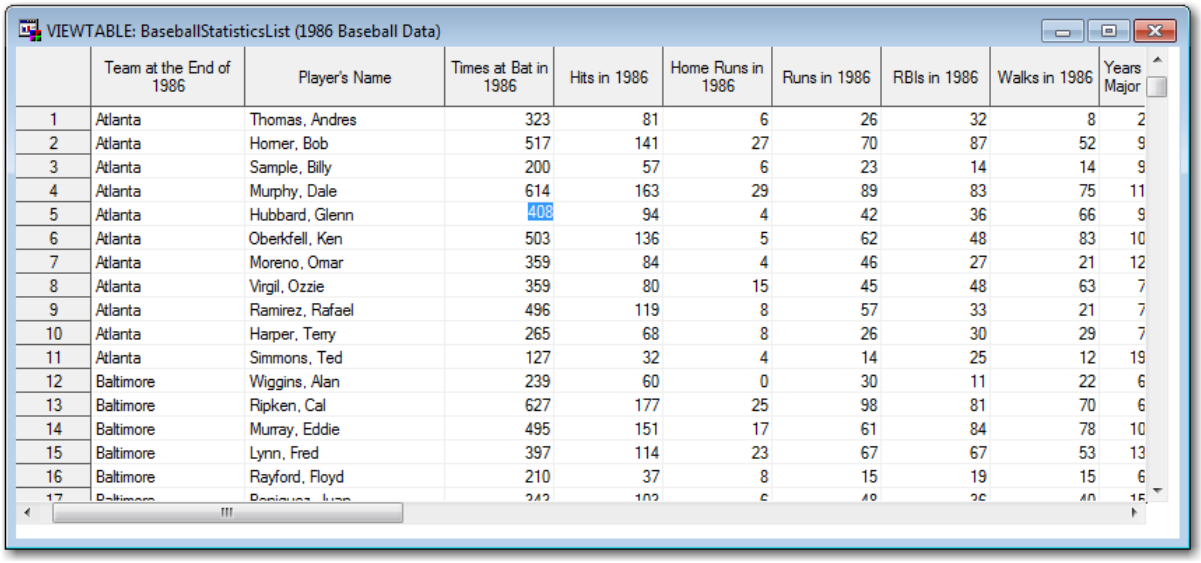
	Team at the End of 1986	Player's Name	Times at Bat in 1986	Hits in 1986	Home Runs in 1986	Runs in 1986	RBIs in 1986	Walks in 1986	Years Major
1	Atlanta	Thomas, Andres	323	81	6	26	32	8	2
2	Atlanta	Horner, Bob	517	141	27	70	87	52	9
3	Atlanta	Sample, Billy	200	57	6	23	14	14	9
4	Atlanta	Murphy, Dale	614	163	29	89	83	75	11
5	Atlanta	Hubbard, Glenn	408	94	4	42	36	66	9
6	Atlanta	Oberkfell, Ken	503	136	5	62	48	83	10
7	Atlanta	Moreno, Omar	359	84	4	46	27	21	12
8	Atlanta	Virgil, Ozzie	359	80	15	45	48	63	7
9	Atlanta	Ramirez, Rafael	496	119	8	57	33	21	7
10	Atlanta	Harper, Terry	265	68	8	26	30	29	7
11	Atlanta	Simmons, Ted	127	32	4	14	25	12	19
12	Baltimore	Wiggins, Alan	239	60	0	30	11	22	6
13	Baltimore	Ripken, Cal	627	177	25	98	81	70	6
14	Baltimore	Murray, Eddie	495	151	17	61	84	78	10
15	Baltimore	Lynn, Fred	397	114	23	67	67	53	13
16	Baltimore	Rayford, Floyd	210	37	8	15	19	15	6
17	Baltimore	Benigno, Juan	242	102	6	48	36	40	15

セル値の編集

デフォルトでは、**VIEWTABLE** に既存のテーブルが参照モード(テーブルデータが保護された状態)で表示されます。テーブルを編集するには、編集モードに切り替える必要があります。編集モードに切り替えるには、次の手順を実行します。

1. テーブルを開いた状態で、**編集**メニューから**編集** ⇨ **編集モード**を選択します。
2. テーブルのセルをクリックすると、セルの値が強調表示されます。

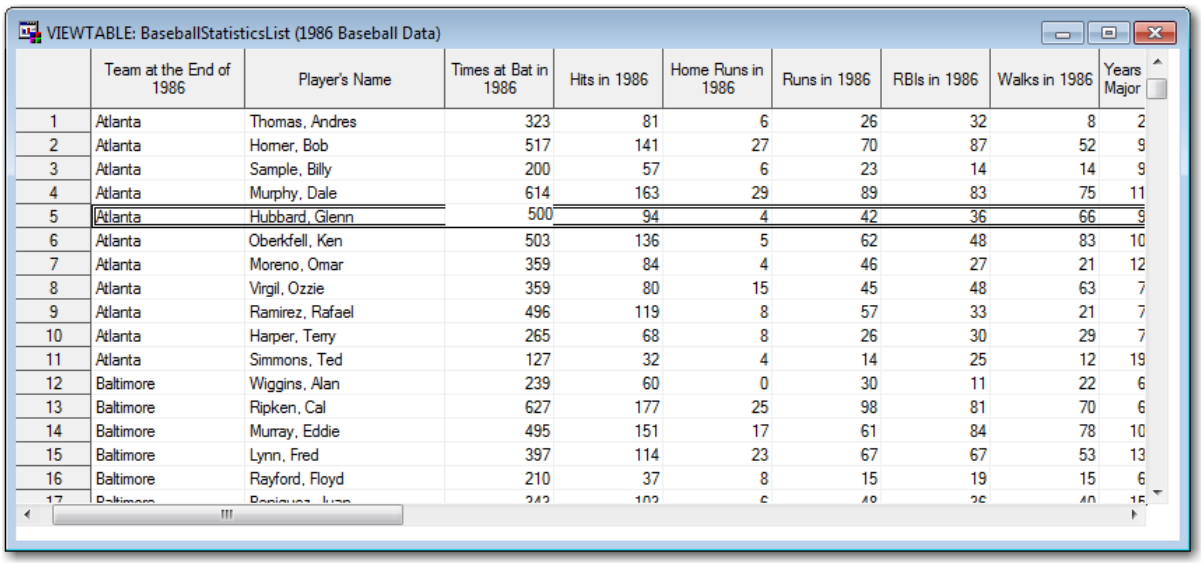
   次の例では、5 番目の行の 3 番目のセルが選択されています。



	Team at the End of 1986	Player's Name	Times at Bat in 1986	Hits in 1986	Home Runs in 1986	Runs in 1986	RBIs in 1986	Walks in 1986	Years Major
1	Atlanta	Thomas, Andres	323	81	6	26	32	8	2
2	Atlanta	Horner, Bob	517	141	27	70	87	52	9
3	Atlanta	Sample, Billy	200	57	6	23	14	14	9
4	Atlanta	Murphy, Dale	614	163	29	89	83	75	11
5	Atlanta	Hubbard, Glenn	408	94	4	42	36	66	9
6	Atlanta	Oberkfell, Ken	503	136	5	62	48	83	10
7	Atlanta	Moreno, Omar	359	84	4	46	27	21	12
8	Atlanta	Virgil, Ozzie	359	80	15	45	48	63	7
9	Atlanta	Ramirez, Rafael	496	119	8	57	33	21	7
10	Atlanta	Harper, Terry	265	68	8	26	30	29	7
11	Atlanta	Simmons, Ted	127	32	4	14	25	12	19
12	Baltimore	Wiggins, Alan	239	60	0	30	11	22	6
13	Baltimore	Ripken, Cal	627	177	25	98	81	70	6
14	Baltimore	Murray, Eddie	495	151	17	61	84	78	10
15	Baltimore	Lynn, Fred	397	114	23	67	67	53	13
16	Baltimore	Rayford, Floyd	210	37	8	15	19	15	6

3. セルの新しい値を入力し、**Enter** を押します。

   次の例では、**Times at Bat in 1986** に対して新しい値でセルが更新されます。



	Team at the End of 1986	Player's Name	Times at Bat in 1986	Hits in 1986	Home Runs in 1986	Runs in 1986	RBIs in 1986	Walks in 1986	Years Major
1	Atlanta	Thomas, Andres	323	81	6	26	32	8	2
2	Atlanta	Horner, Bob	517	141	27	70	87	52	9
3	Atlanta	Sample, Billy	200	57	6	23	14	14	9
4	Atlanta	Murphy, Dale	614	163	29	89	83	75	11
5	Atlanta	Hubbard, Glenn	500	94	4	42	36	66	9
6	Atlanta	Oberkfell, Ken	503	136	5	62	48	83	10
7	Atlanta	Moreno, Omar	359	84	4	46	27	21	12
8	Atlanta	Virgil, Ozzie	359	80	15	45	48	63	7
9	Atlanta	Ramirez, Rafael	496	119	8	57	33	21	7
10	Atlanta	Harper, Terry	265	68	8	26	30	29	7
11	Atlanta	Simmons, Ted	127	32	4	14	25	12	19
12	Baltimore	Wiggins, Alan	239	60	0	30	11	22	6
13	Baltimore	Ripken, Cal	627	177	25	98	81	70	6
14	Baltimore	Murray, Eddie	495	151	17	61	84	78	10
15	Baltimore	Lynn, Fred	397	114	23	67	67	53	13
16	Baltimore	Rayford, Floyd	210	37	8	15	19	15	6

4. **ファイル**メニューから、**ファイル** ⇨ **閉じる**を選択します。
5. 保留中の変更をテーブルに保存するように要求するメッセージが表示されたら、変更を保存する場合は**はい**、変更を破棄するには**いいえ**をクリックします。

注: ある行を変更し、別の行のセルを変更した場合、最初の行の変更が自動的に保存されます。**ファイル** ⇨ **閉じる**を選択すると、保留中の変更を 2 番目の行に保存するように要求されます。

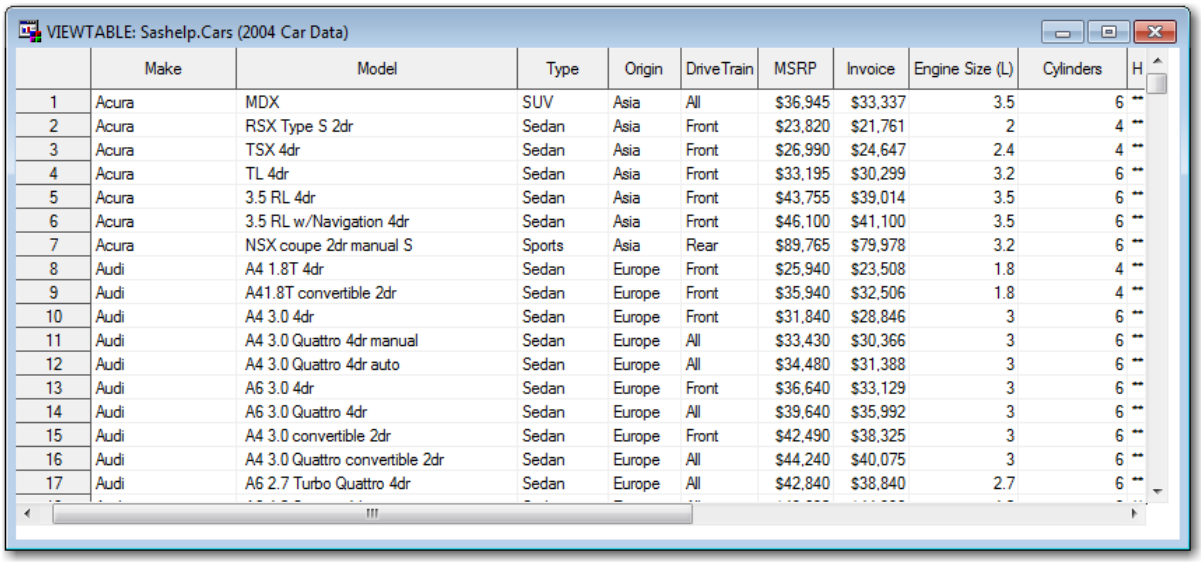
WHERE 式を用いたデータのサブセット化

テーブルの行のサブセット化

VIEWTABLE ウィンドウでは、1 つ以上の条件を満たす行のみを表示するように表示をサブセット化できます。テーブルの行を部分表示するには、次の操作を実行します。

1. **エクスプローラ**ウィンドウで、ライブラリを開き、サブセットを作成するテーブルをダブルクリックします。

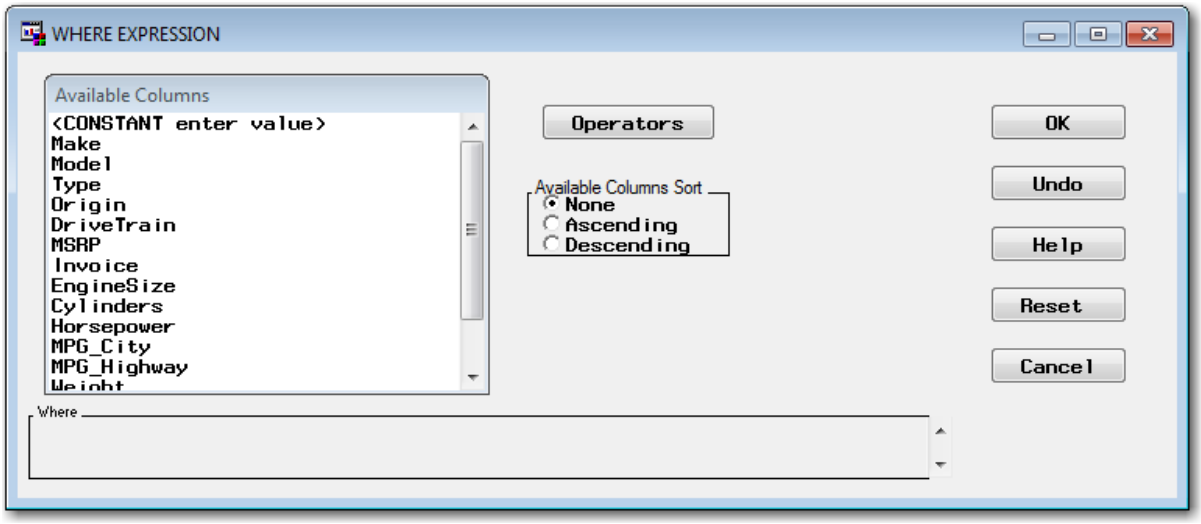
   次の例では、Cars データテーブルを選択しています。



	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	Engine Size (L)	Cylinders	H
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2	4	
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3	6	
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	$33,430	$30,366	3	6	
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	$34,480	$31,388	3	6	
13	Audi	A6 3.0 4dr	Sedan	Europe	Front	$36,640	$33,129	3	6	
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	$39,640	$35,992	3	6	
15	Audi	A4 3.0 convertible 2dr	Sedan	Europe	Front	$42,490	$38,325	3	6	
16	Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	All	$44,240	$40,075	3	6	
17	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	All	$42,840	$38,840	2.7	6	

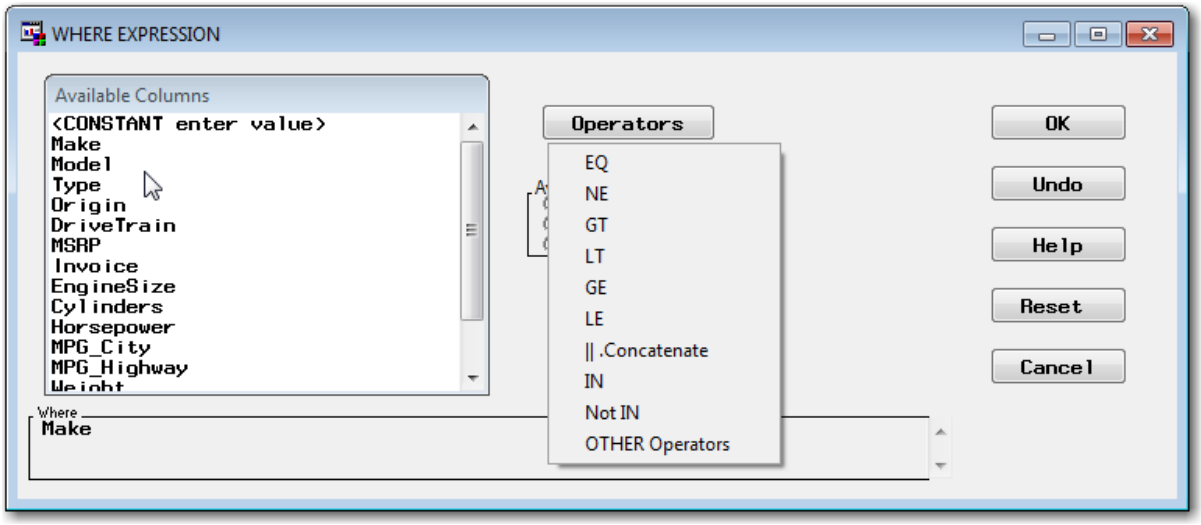
2. 見出しではないテーブルのセルを右クリックし、メニューから **Where** を選択します。

WHERE 式ウィンドウが表示されます。



3. **利用可能な列**リストで、列を選択し、**演算子**メニューから演算子を選択します。

   次の例では、**Make** を**選択可能な列**リストで選択し、**EQ** (等しい)を**演算子**メニューから選択しています。WHERE 式がウィンドウ下部の **Where** ボックスで構築されています。

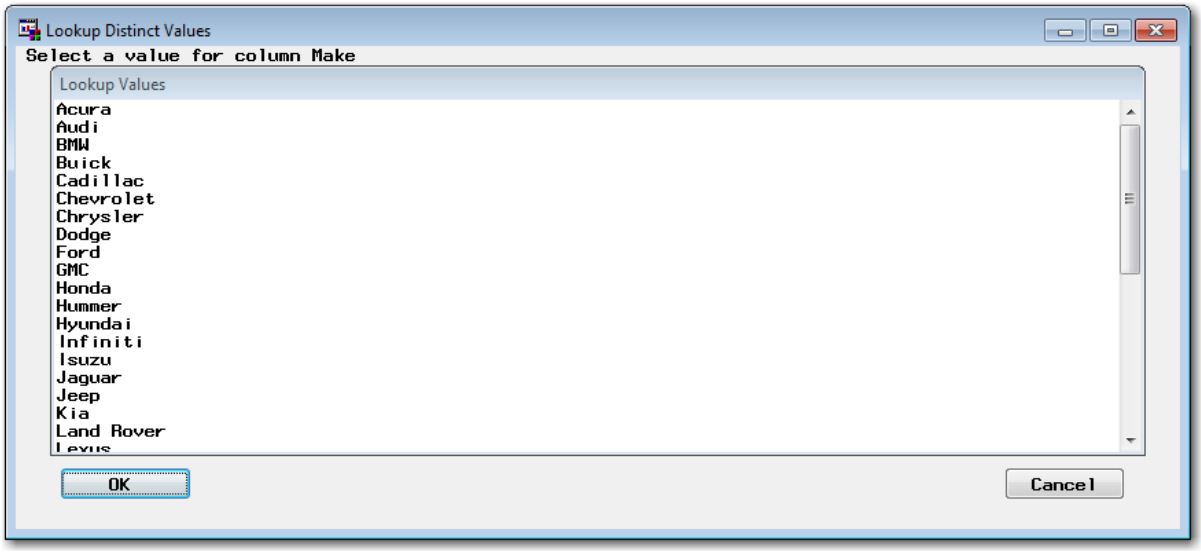


4. **利用可能な列**リストで、別の値を選択して WHERE 式を完成させます。

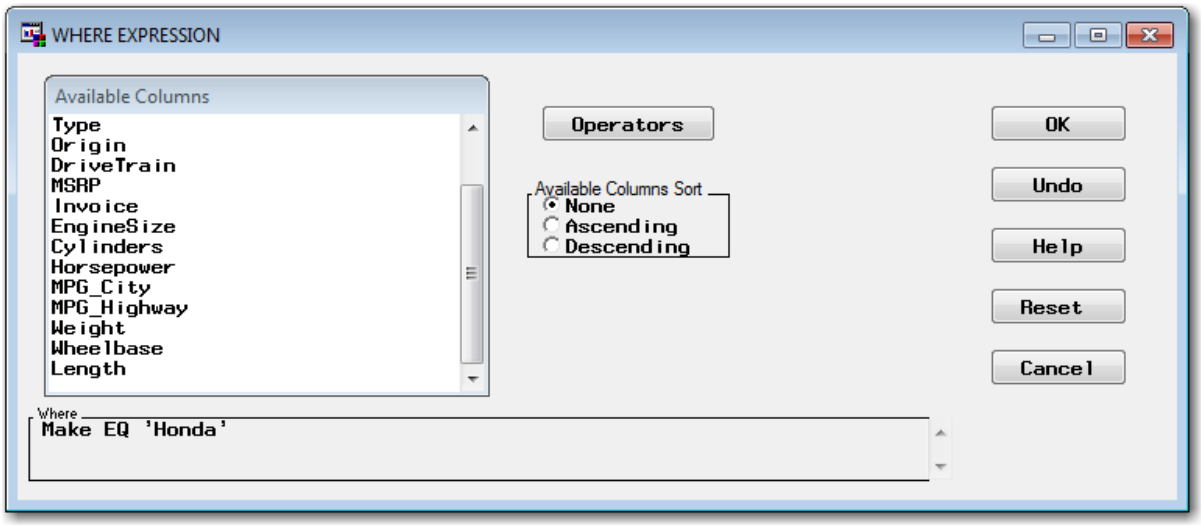
   次の例では、**選択可能な列**ウィンドウの最下部までスクロールし、**<LOOKUP distinct values>**を選択します。

5. 表示された**定数値の参照**ウィンドウで、値を選択します。

次の例では、**Honda** を選択しています。

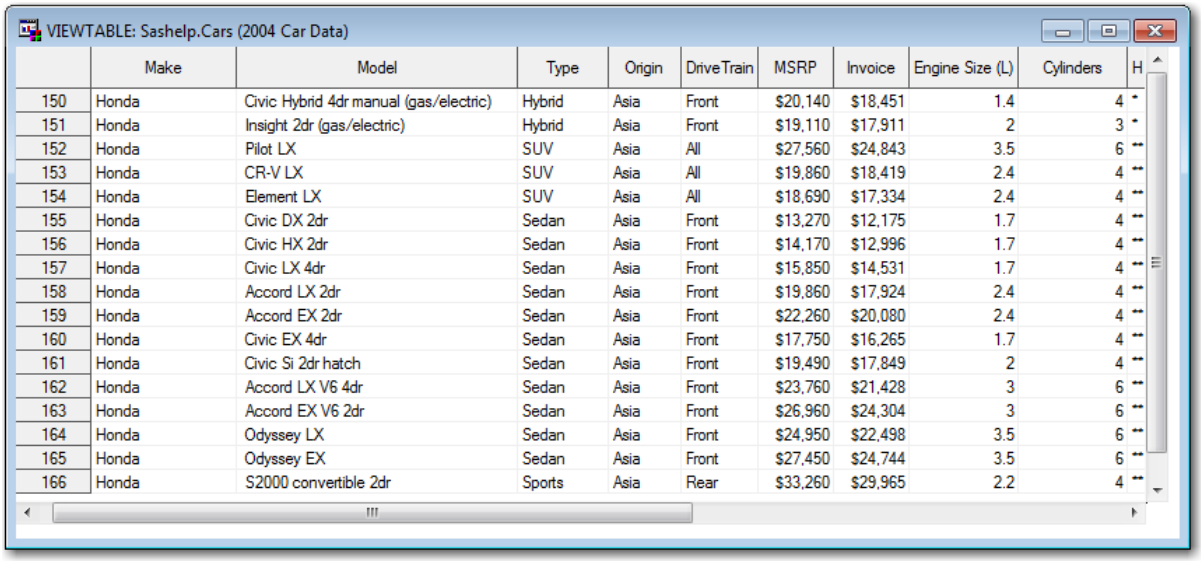


完成した WHERE 式がウィンドウ下部の **Where** ボックスに表示されます。



6. **OK** をクリックし、**WHERE 式**ウィンドウを閉じます。

次の例では、**Make** の値が Honda である行のみが **VIEWTABLE** に表示されます。



VIEWTABLE: Sashelp.Cars (2004 Car Data)

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	Engine Size (L)	Cylinders	H
150	Honda	Civic Hybrid 4dr manual (gas/electric)	Hybrid	Asia	Front	$20,140	$18,451	1.4	4	
151	Honda	Insight 2dr (gas/electric)	Hybrid	Asia	Front	$19,110	$17,911	2	3	
152	Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5	6	
153	Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4	4	
154	Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4	4	
155	Honda	Civic DX 2dr	Sedan	Asia	Front	$13,270	$12,175	1.7	4	
156	Honda	Civic HX 2dr	Sedan	Asia	Front	$14,170	$12,996	1.7	4	
157	Honda	Civic LX 4dr	Sedan	Asia	Front	$15,850	$14,531	1.7	4	
158	Honda	Accord LX 2dr	Sedan	Asia	Front	$19,860	$17,924	2.4	4	
159	Honda	Accord EX 2dr	Sedan	Asia	Front	$22,260	$20,080	2.4	4	
160	Honda	Civic EX 4dr	Sedan	Asia	Front	$17,750	$16,265	1.7	4	
161	Honda	Civic Si 2dr hatch	Sedan	Asia	Front	$19,490	$17,849	2	4	
162	Honda	Accord LX V6 4dr	Sedan	Asia	Front	$23,760	$21,428	3	6	
163	Honda	Accord EX V6 2dr	Sedan	Asia	Front	$26,960	$24,304	3	6	
164	Honda	Odyssey LX	Sedan	Asia	Front	$24,950	$22,498	3.5	6	
165	Honda	Odyssey EX	Sedan	Asia	Front	$27,450	$24,744	3.5	6	
166	Honda	S2000 convertible 2dr	Sports	Asia	Rear	$33,260	$29,965	2.2	4	

WHERE 式のクリア

データのサブセット化に使用した WHERE 式をクリアし、テーブルの全データを再表示できます。これには、次の操作を実行します。

1. 列の見出しを除くテーブルの任意の場所を右クリックします。
2. メニューから **WHERE のクリア**を選択します。

   VIEWTABLE ウィンドウで、WHERE 式で作成されたデータの既存のサブセットが削除され、テーブルのすべての行が表示されます。

データのサブセットのエクスポート

データのエクスポートの概要

エクスポートウィザードでは、SAS データセットからデータを読み取り、外部ファイルに書き込みます。SAS データをさまざまな形式にエクスポートできます。使用可能な形式は、動作環境とインストールした SAS 製品によって異なります。

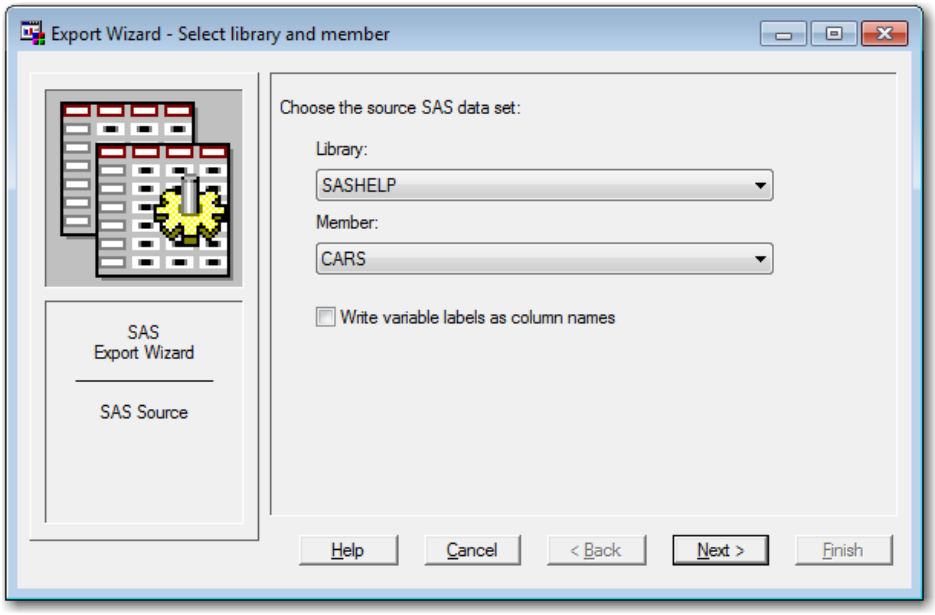
データのエクスポート

データをエクスポートするには、次の操作を実行します。

1. **エクスプローラ**ウィンドウをアクティブにし、**ファイル** ⇨ **データのエクスポート**を選択します。

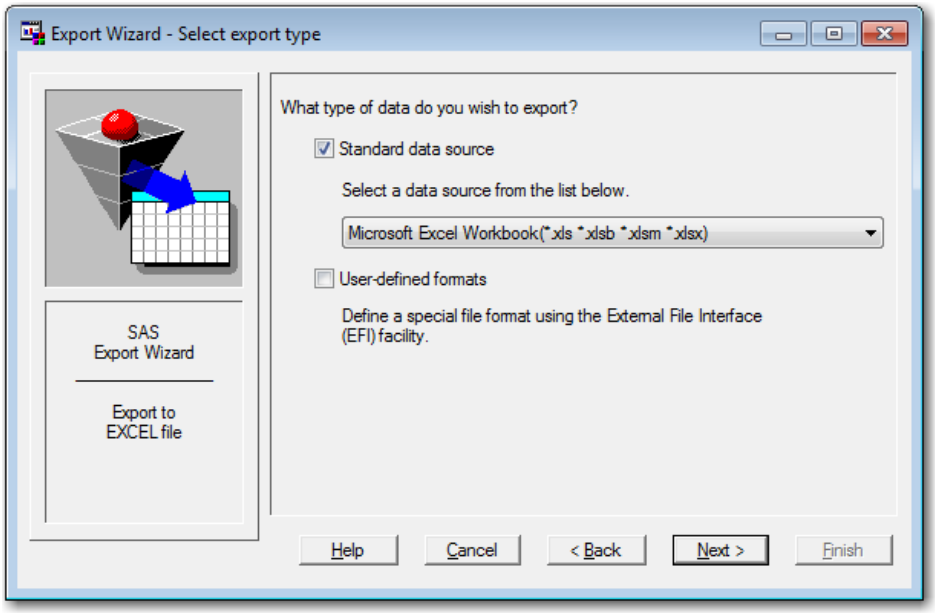
   エクスポートウィザード - ライブラリとメンバの選択ウィンドウが表示されます。
2. データのエクスポート元の SAS データセットを選択します。

次の例では、**Sashelp** がライブラリとして選択され、**Cars** がメンバ名です。



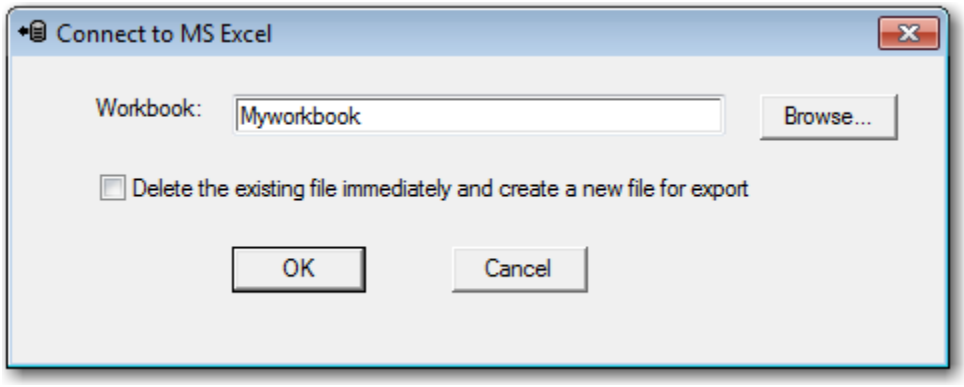
3. **次へ**をクリックすると、**エクスポートウィザード - エクスポートタイプの選択**ウィンドウが表示されます。
4. ファイルのエクスポート先となるデータソースの種類を選択します。

   次の例では、**Microsoft Excel Workbook** を選択しています。デフォルトでは、**標準データソース**が選択されます。



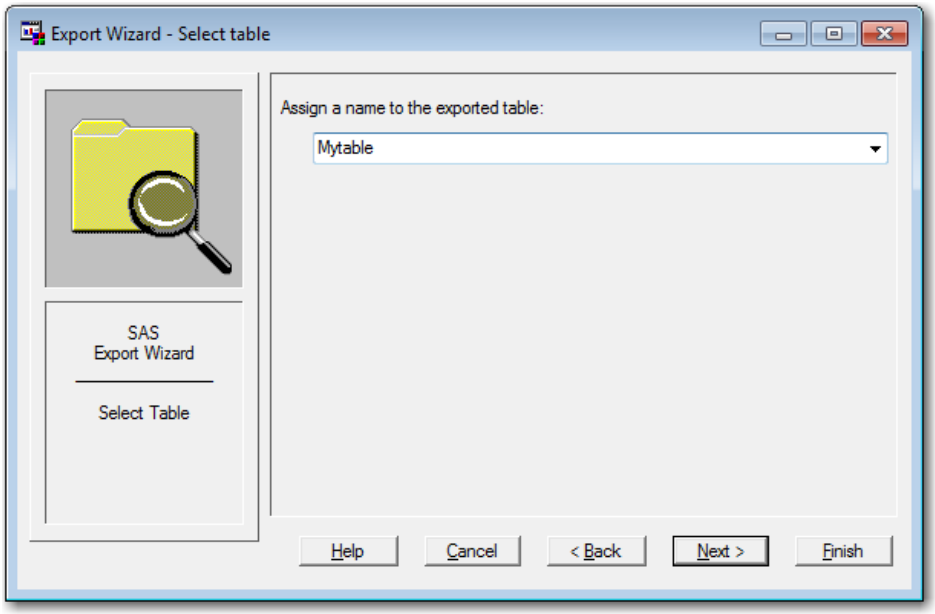
5. **次へ**をクリックし、**MS Excel への接続**ウィンドウを表示します。
6. **ワークブック**フィールドでは、エクスポートしたファイルを格納するワークブックの名前を入力し、**OK** をクリックします。

次の例では、ワークブックの名前として **Myworkbook** を入力しています。



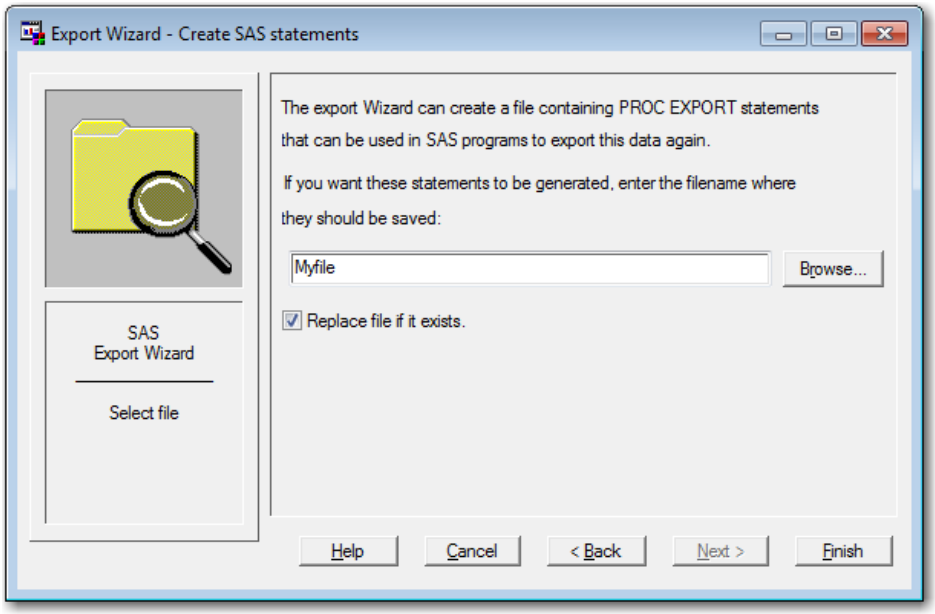
7. **エクスポートウィザード - テーブルの選択**ウィンドウが表示されたら、エクスポートするテーブルの名前を入力します。

   次の例では、**Mytable** がテーブルの名前です。



8. **次へ**をクリックします。
9. PROC EXPORT ステートメントのファイルを後で使用するために作成する場合は、SAS ステートメントを格納するファイルの名前を入力します。

   次の例では、PROC EXPORT ステートメントをファイルに保存します。**既存のファイルを置き換える**チェックボックスをオンにします。



10. **完了**をクリックするとこのタスクを終了します。

データのテーブルへのインポート

データのインポートの概要

データが標準ファイル形式で格納された場合でも、独自の特殊なファイル形式で格納された場合でも、インポートウィザードを使用すると、データを SAS のテーブルにインポートできます。インポートできるファイルの種類は、動作環境によって異なります。

標準ファイルのインポート

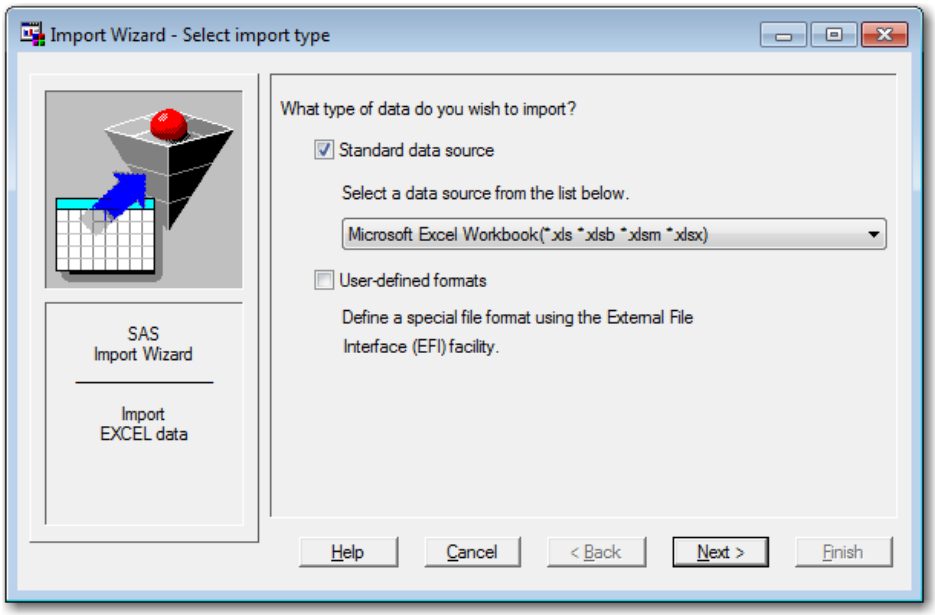
標準ファイルをインポートするには、次の操作を実行します。

1. **エクスプローラ**ウィンドウをアクティブにし、**ファイル** ⇨ **データのインポート**を選択します。

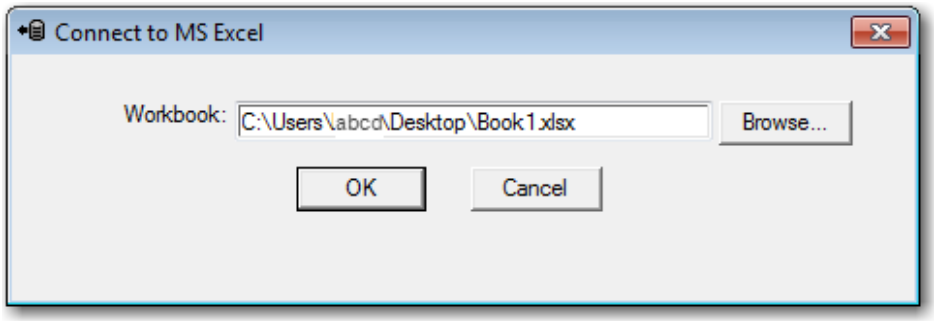
   インポートウィザード - インポートタイプの選択ウィンドウが表示されます。

2. **データソースの選択**メニューからデータソースを選択すると、インポートするファイルの種類を選択します。

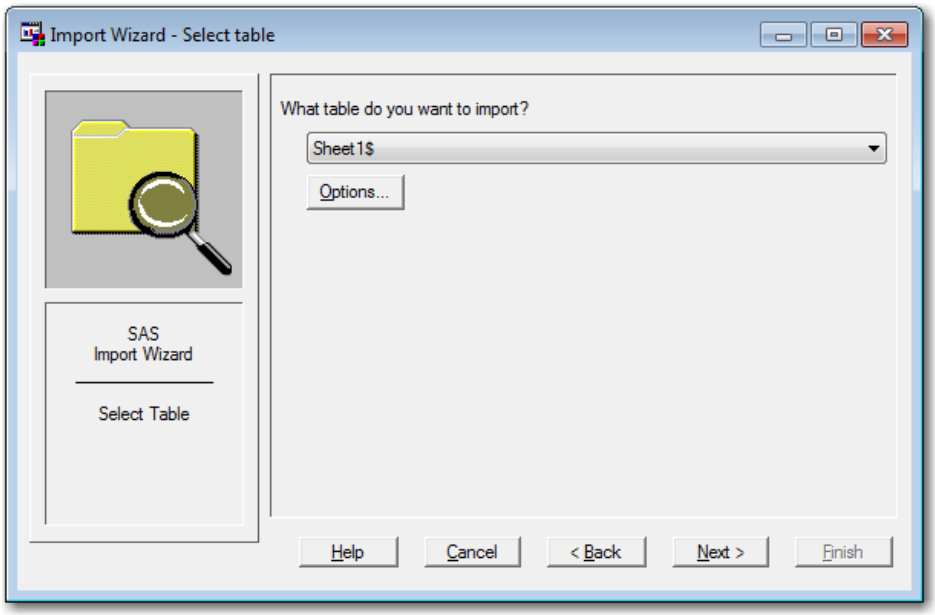
   デフォルトでは、**標準データソース**が選択されます。次の例では、**Microsoft Excel Workbook** を選択しています。



3. **次へ**をクリックすると処理を続行します。
4. **MS Excel への接続**ウィンドウで、エクスポートするファイルのパス名を入力し、**OK**をクリックします。

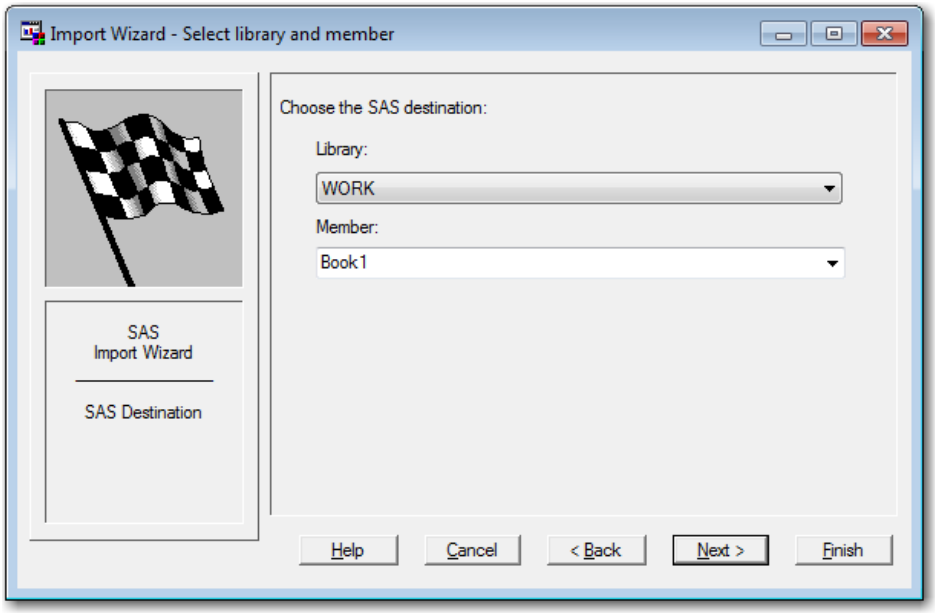


5. **インポートウィザード - テーブルの選択**ウィンドウで、インポートするテーブルの名前を入力します。

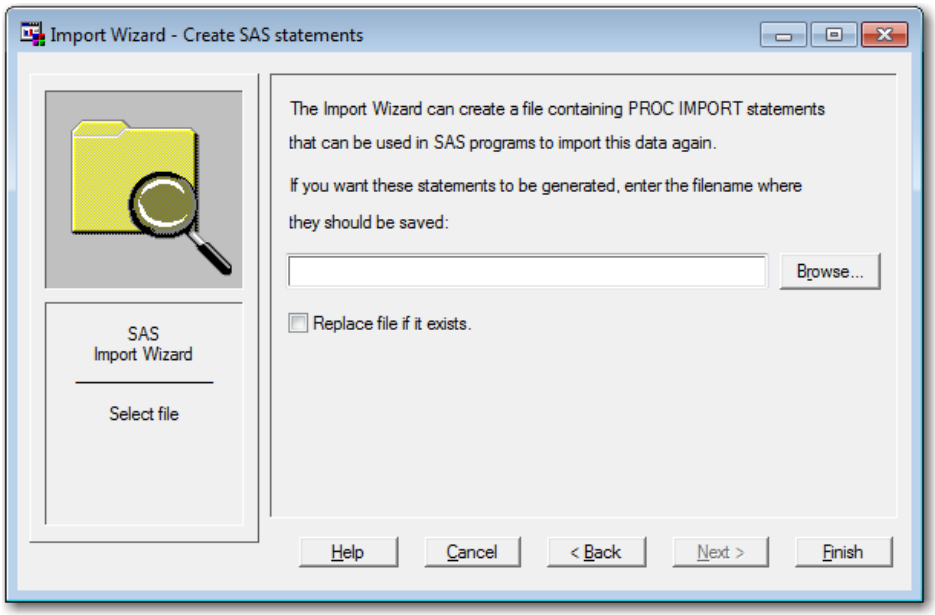


6. **次へ**をクリックすると処理を続行します。
7. **インポートウィザード - ライブラリとメンバの選択**ウィンドウで、インポートしたファイルを格納する場所を入力します。

   次の例では、**Work** がライブラリとして選択され、**Book1** がメンバ名として選択されます。



8. **次へ**をクリックすると処理を続行します。
9. PROC IMPORT ステートメントのファイルを後で使用するために作成する場合は、SAS ステートメントを格納するファイルの名前を入力します。



10. **完了**をクリックするとこのタスクを終了します。

非標準ファイルのインポート

データが標準形式ではない場合、EFI (External File Interface)機能を使用すると、データをインポートできます。このツールでは、ファイル形式を定義し、さまざまなフォーマットオプションを利用できます。EFI を使用するには、インポートウィザードのユーザー定義のファイル形式を選択し、データファイルを記述する手順に従います。

3 部

DATA ステップの概念

18 章

DATA ステップの処理

DATA ステップの特徴

Base SAS ソフトウェアを使用して SAS データセットを作成する場合、DATA ステップを使用する方法が最も一般的です。DATA ステップとは、DATA ステートメントで始まる

SAS 言語ステートメントの集まりです。言語ステートメントの集まりには、既存の SAS データセットを操作したり、生データファイルから SAS データセットを生成したりする他のプログラムステートメントが含まれています。

DATA ステップを使うと次のようなタスクを実行できます。

- SAS データセット(SAS データファイルまたは SAS ビュー)の作成
- 生データを含んでいる入力ファイル(外部ファイル)からの SAS データセットの作成
- 既存の SAS データセットのサブセット化、マージ、変更、更新などの操作による、新しい SAS データセットの作成
- データの分析、操作、プレゼンテーション
- 新しい変数の値の計算
- レポートの作成、またはディスクやテープへのファイルの書き出し
- 情報の取り出し
- ファイル管理

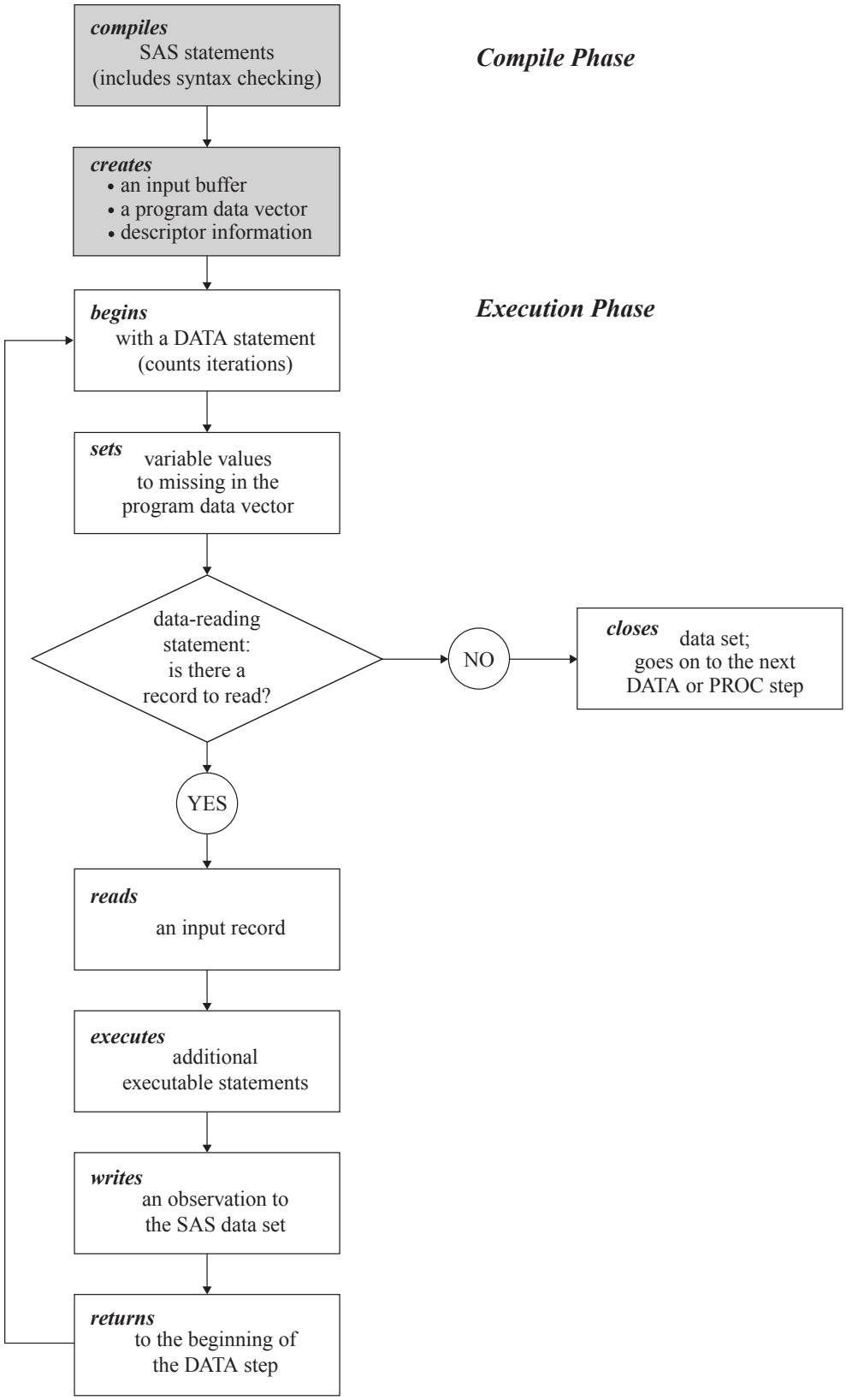
注: DATA ステップを実行すると、SAS データセットが作成されます。作成されるデータセットは、SAS データファイルまたは SAS ビューのどちらかになります。SAS データファイルは実際のデータ値を格納するものであり、SAS ビューはデータの取り出しや処理を行うための情報を格納するものです。SAS ビューは、ほとんどの場合 SAS データファイルとして使用できます。そのため、本書ではより意味の広い SAS データセットという用語を使用します。

DATA ステッププロセスの概要

処理フロー

DATA ステップを実行しようとしてサブミットすると、まずコンパイルされ、その後に実行されます。次の図は、典型的な SAS DATA ステップにおける処理フローを示しています。

図 18.1 DATA ステップにおける処理フロー



コンパイルフェーズ

DATA ステップを実行しようとしてサブミットすると、SAS ステートメントの構文が確認され、ステートメントがコンパイルされます。つまり、SAS ステートメントが自動的にマシンコードに変換されます。このフェーズでは、新しい変数の種類と長さが特定されるほか、その変数を後で参照するための変数の種類の変換が必要かどうかが決定されます。コンパイル時には、次のものが作成されます。

入力バッファ
: 一時的にデータを格納するメモリ領域です。INPUT ステートメントの実行時に、生データの各レコードが読み込まれます。入力バッファは、DATA ステップが生データを読み込む場合にのみ作成されます。(DATA ステップが SAS データセットを読み込む場合、データは直にプログラムデータベクトルへと読み込まれます。)

プログラムデータベクトル(PDV)
: メモリ上の論理領域であり、ここで SAS System はデータセット(オブザベーション)を 1 つずつ作成します。プログラムを実行すると、入力バッファからデータ値が読み込まれるか、SAS 言語ステートメントが実行されてデータ値が作成されます。そのデータ値が、プログラムデータベクトル(PDV)内の適切な変数に割り当てられます。SAS System は、このベクトルにある値を 1 つのオブザベーションとして SAS データセットに書き出します。

  PDV には、データセット変数と計算済みの変数のほかに、_N_と_ERROR_という自動変数が格納されます。自動変数_N_は、DATA ステップが反復された回数をカウントします。自動変数_ERROR_は、DATA ステップの実行中にエラーが発生したかどうかを示す値です。_ERROR_の値は 0 または 1 のどちらかであり、0 はエラーが発生しなかったことを、1 はエラーが 1 回以上発生したことを示します。これらの自動変数は、出力データセットには書き出されません。

ディスクリプタ情報
: SAS System が SAS データセットごとに作成し、保持している情報です。データセットの属性と変数の属性が含まれます。たとえば、この情報には、データセットの名前とメンバタイプ、データセットの作成日時、変数の数、変数名、変数のデータ型(文字または数値)などが含まれます。ディスクリプタ情報には、(データセットに定義されている場合は)拡張属性についての情報も含まれます。拡張属性ディスクリプタ情報には、属性名、変数名および属性値が含まれます。

実行フェーズ

デフォルトでは、単純な DATA ステップは作成されるオブザベーションごとに 1 回ずつ反復されます。単純な DATA ステップの実行時の処理フローは次のようになります。

1. DATA ステップは DATA ステートメントで始まります。DATA ステートメントを実行するたびに、新しい DATA ステップの反復が開始され、自動変数_N_の値が 1 つ増分します。
2. プログラムデータベクトル(PDV)内に新しく作成されたプログラム変数に、欠損値が設定されます。
3. 生データファイルからデータレコードが読み込まれ、入力バッファに格納されます。または、SAS データセットからオブザベーションが直接読み込まれ、プログラムデータベクトルに格納されます。レコードの読み込みには、INPUT、MERGE、SET、MODIFY、UPDATE ステートメントを使用できます。
4. 現在のレコードを処理するための後続プログラムステートメントがある場合には、それらが実行されます。

5. ステートメントの実行がすべて終了すると、結果の出力、先頭への復帰、値のリセットが自動的に実行されます。SAS データセットにオブザベーションが書き出され、処理は自動的に DATA ステップの先頭に戻ります。プログラムデータベクトル内にある、INPUT ステートメントと割り当てステートメントによって作成された値は、欠損値にリセットされます。SET、MERGE、MODIFY、UPDATE ステートメントを使用して読み込んだ変数は、ここでは欠損値にリセットされません。
6. SAS データセットまたは生データでファイルの終端が検出されていない場合、次のレコードまたはオブザベーションが読み込まれ、現在のオブザベーションを処理するための後続のプログラムステートメントが実行されます。
7. SAS データセットまたは生データファイルでファイルの終端が検出されると、DATA ステップが終了します。

注: この図は、DATA ステップでのデフォルトの処理を示しています。プログラム内には、INPUT ステートメントや SET ステートメントなどのデータ読み込みステートメント、OUTPUT ステートメントなどのデータ書き出しステートメントを任意の順序で記述できます。

DATA ステップの処理:実例を通じたステップごとの説明

DATA ステップの例

次のステートメントは、生データを読み込んで合計を計算し、データセットを作成する DATA ステップの例です。

```
data total_points (drop=TeamName);  1
   input TeamName $ ParticipantName $ Event1 Event2 Event3;  2
   TeamTotal + (Event1 + Event2 + Event3);  3
   datalines;
Knights Sue    6  8  8
Kings Jane     9  7  8
Knights John   7  7  7
Knights Lisa   8  9  9
Knights Fran   7  6  6
Knights Walter 9  8 10
;

proc print data=total_points;
run;
```

1 DROP=データセットオプションでは、変数 TeamName が出力 SAS データセット Total_Points に書き出されないように除外します。

2 INPUT ステートメントでは、各変数について変数名、データ型(文字または数値)、データレコード内での相対位置を指定することによって、データを記述します。

3 SUM ステートメントでは、3 つのイベントのスコアを集計した値を変数 TeamTotal に格納します。

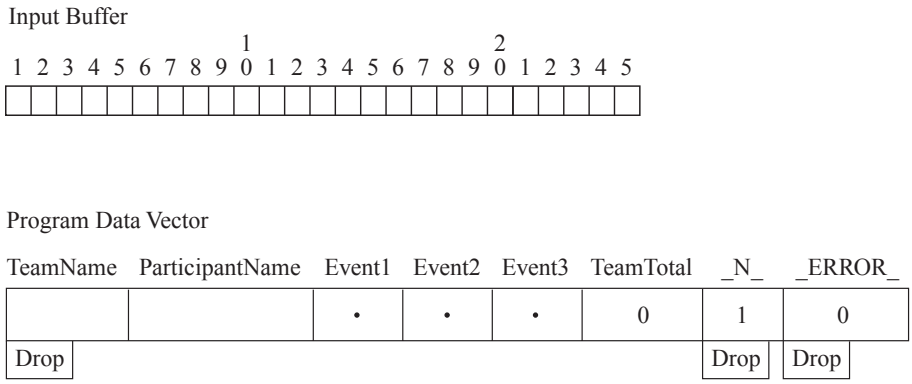
入力バッファとプログラムデータベクトルの作成

DATA ステップステートメントがコンパイルされると、SAS System は入力バッファを作成するかどうかを決定します。前述の例のように、入力ファイルから生データを読み込

む場合、SAS System は、プログラムデータベクトル(PDV)にデータを移動する前に、データを保持するための入力バッファを作成します。(入力バッファは、入力ファイルが SAS データセットである場合は作成されません。この場合、入力データは PDV に直接書き出されます。)

PDV には、入力データセットにあるすべての変数、DATA ステップステートメントによって作成された変数、および_N_と_ERROR_という 2 つの変数が格納されます。変数_N_と変数_ERROR_は、すべての DATA ステップに 1 つずつ、自動的に生成されます。自動変数_N_は、DATA ステップの反復が開始された回数を表しています。自動変数_ERROR_は、バイナリスイッチのように機能する値です。DATA ステップ内にエラーがまったくない場合は 0 になり、エラーが 1 つでもある場合は 1 になります。DATA ステップのコンパイルが終了した時点での入力バッファとプログラムデータベクトルを次の図に示します。

図 18.2 *入力バッファとプログラムデータベクトルの作成*



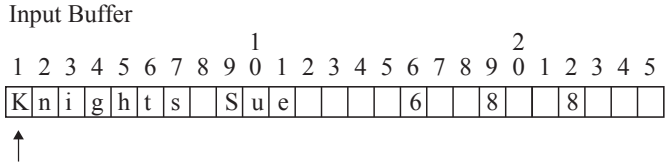
INPUT ステートメントと合計ステートメントによって作成された変数(TeamName、ParticipantName、Event1、Event2、Event3、TeamTotal)は、まず欠損値として設定されます。この図では、数値変数をピリオド(.)に初期化し、文字変数をブランクに初期化しています。自動変数_N_には 1 が設定され、自動変数_ERROR_には 0 が設定されます。

PDV にある変数 TeamName には、DATA ステートメントで DROP=データセットオプションが設定されているため Drop という印が付けられています。除外された変数は、SAS データセットには書き出されません。DATA ステップによって作成された自動変数は SAS データセットに書き出されないため、自動変数_N_と_ERROR_も除外されます。自動変数の詳細については、4 章, “SAS 変数” (35 ページ) を参照してください。

レコードの読み込み

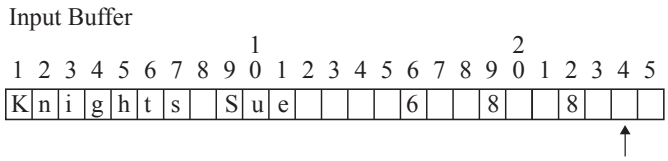
SAS System が、最初のデータ行を入力バッファに読み込むと、入力バッファの先頭には、入力ポインタが配置されます。このポインタは、SAS System がデータを入力バッファから読み込む際に使用するものであり、データの位置を保持してデータレコードを読み込めるようにします。SAS System がデータを読み込む前の入力バッファ内での入力ポインタの位置を下記の図に示します。

図 18.3 *データを読み込む前の入力バッファ内でのポインタの位置*



INPUT ステートメントは入力バッファ内のレコードからデータ値を読み込み、そのデータ値を PDV に書き出します。PDV に書き出されたデータ値は、変数値になります。次の図は、SAS System が最初のレコードを読み込んだ後の、入力バッファ内での入力ポインタの位置、および PDV に格納された値を示しています。

図 18.4 *最初のレコードが読み込まれた後のプログラム データベクトル*



Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
Knights	Sue	6	8	8	0	1	0
Drop						Drop	Drop

INPUT ステートメントにより各変数の値が読み込まれた後に、合計ステートメントが実行されます。変数 TeamTotal の値が計算され、その値が PDV に書き出されます。次の図は、SAS System がデータセットにオブザベーションを書き出す前の値がすべて格納された PDV を示しています。

図 18.5 *合計ステートメントで計算された値が格納されたプログラム データベクトル*

Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
Knights	Sue	6	8	8	22	1	0
Drop						Drop	Drop

SAS データセットへのオブザベーションの書き出し

DATA ステップの最後のステートメントが実行されると、PDV にあるすべての値が、データセット Total_Points に 1 つのオブザベーションとして書き出されます。ただし、除外対象に指定された値は書き出されません。次の図は、データセット Total_Points に書き出される最初のオブザベーションを示しています。

図 18.6 *データセット Total_Points に書き出される最初のオブザベーション*

Output SAS Data Set TOTAL_POINTS: 1st observation

ParticipantName	Event1	Event2	Event3	TeamTotal
Sue	6	8	8	22

DATA ステートメントに戻って、次の反復を開始します。PDV に格納されている値は、次の規則に従ってリセットされます。

- INPUT ステートメントによって作成された変数の値は、欠損値に設定されます。
- 合計ステートメントによって作成された値は、自動的に保持されます。

- 自動変数_N_の値は 1 つ増分され、_ERROR_の値は 0 にリセットされます。

次の図は、この時点で PDV に格納されている値を示しています。

図 18.7 *プログラムデータベクトル内の現在の値*

Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
		•	•	•	22	2	0
Drop						Drop	Drop

次のレコードの読み込み

SAS System は、次のレコードを入力バッファに読み込みます。INPUT ステートメントによって入力バッファからデータ値が読み込まれ、そのデータ値が PDV に書き出されます。合計ステートメントによって、Event1、Event2、Event3 の値が変数 TeamTotal に加算されます。自動変数_N_の 2 という値は、DATA ステップの次の反復が開始されたことを示しています。次の図は、入力バッファ、2 番目のレコードの PDV、および最初の 2 つのオブザベーションを保持した SAS データセットを示しています。

図 18.8 *入力バッファ、プログラムデータベクトル、および最初の 2 つのオブザベーション*

Input Buffer

```
                    1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
C a r d i n a l s   J a n e   9     7     8
                                          ↑
```

Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
Cardinals	Jane	9	7	8	46	2	0
Drop						Drop	Drop

Output SAS Data Set TOTAL_POINTS: 1st and 2nd observations

ParticipantName	Event1	Event2	Event3	TeamTotal
Sue	6	8	8	22
Jane	9	7	8	46

SAS System がレコードの読み込みを続けると、より多くの参加者のスコアが加算されていき、変数 TeamTotal の値が大きくなっていきます。自動変数_N_は、DATA ステップの反復が開始されるたびに増分されます。このプロセスは、SAS System が入力ファイルの終端に到達するまで続けられます。

DATA ステップの実行の終了

DATA ステップは、最後の入力レコードを処理すると実行を停止します。Total_Points データセットにある出力結果は、PROC PRINT を使用して出力できます。

```
data total_points (drop=TeamName);
   input TeamName $ ParticipantName $ Event1 Event2 Event3;
   TeamTotal + (Event1 + Event2 + Event3);
   datalines;
Knights Sue    6  8  8
Cardinals Jane 9  7  8
Knights John   7  7  7
Cardinals Lisa 8  9  9
Cardinals Fran 7  6  6
Knights Walter 9  8 10
;
proc print data=total_points;
   title 'Total Team Scores';
run;
```

アウトプット 18.1 *サンプルの DATA ステップからの出力*

Total Team Scores

Obs	ParticipantName	Event1	Event2	Event3	TeamTotal
1	Sue	6	8	8	22
2	Jane	9	7	8	46
3	John	7	7	7	67
4	Lisa	8	9	9	93
5	Fran	7	6	6	112
6	Walter	9	8	10	139

DATA ステップの実行について

DATA ステップのデフォルトの実行順序

次の表は、DATA ステップにおけるステートメントのデフォルトの実行順序を示しています。DATA ステップは、DATA ステートメントで開始します。DATA ステップでは、通常、1 つ以上の SAS データセットを作成するように指定します。(出力データセットを作成しない場合は、データセット名としてキーワード_NULL_を使用します。)オプションのプログラムステートメントを記述すると、データを処理できます。オブザベーションを処理した最後には、デフォルトの処理が実行されます。

表 18.1 *DATA ステップにおける SAS ステートメントのデフォルト実行順序*

DATA ステップの構造	アクション
DATA ステートメント	ステップを開始します。 反復回数をカウントします。
データ読み込みステートメント: *	
INPUT	生データソースから入力データレコードのデータ値を読み込み、対応する変数に割り当てます。
SET	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。
MERGE	複数の SAS データセットからオブザベーションを読み込み、結合して 1 つのオブザベーションにします。
MODIFY	既存の SAS データセットにあるオブザベーションの置換、削除、追加を行います。
UPDATE	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。
オプションの SAS プログラムステートメント (次はその例):	現在のオブザベーションが持つ値をさらに処理します。
FirstQuarter=Jan+Feb+Mar; if RetailPrice < 500;	現在のオブザベーションにある変数 FirstQuarter の値を計算します。 現在のオブザベーションを変数 RetailPrice の値を使用してサブセット化します。
オブザベーション処理の最後に実行されるデフォルト処理	
DATA ステップの最後: 結果の自動出力、DATA ステップの先頭への自動復帰 DATA ステップの先頭: 自動リセット	オブザベーションを SAS データセットに書き出します。 DATA ステートメントの先頭に戻ります。 プログラムデータベクトル内の値を欠損値にリセットします。

* この表は、DATA ステップでのデフォルトの処理を示しています。DATA ステップ内にあるステートメントの順序は変更できます。データ読み込みステートメントの前にプログラムステートメントを記述して、定数の再初期化などを実行することもできます。

注: データの読み込みと操作は、SAS 関数を使用して実行することもできます。ステートメントや関数を使用してデータを処理する方法については、see “Using Functions to Manipulate Files” (*SAS Functions and CALL Routines: Reference*)を参照してください。SAS 関数の詳細については、“SAS Functions and CALL Routines by Category” (*SAS Functions and CALL Routines: Reference*)の SAS 入出力ファイルと外部ファイルの説明を参照してください。

デフォルトの実行順序の変更

ステートメントを使用したデフォルトの実行順序の変更

DATA ステップにおけるデフォルトの実行順序を変更して、プログラムの動作を制御できます。SAS 言語ステートメントにはさまざまな柔軟性があり、DATA ステップ内でのデフォルトの実行順序を変更できます。次のリストは、DATA ステッププログラム内での実行順序を制御するための一般的な方法を示しています。

表 18.2 実行順序を変更するための一般的な方法

タスク	考えられる方法
レコードの読み込み	データセットのマージ、変更、結合を行います。 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します。 処理対象のレコードをランダムに選択します。 複数の外部ファイルから読み込みます。 ステートメントオプションまたはデータセットオプションを使用して、レコードから特定のデータフィールドを読み込みます。
データの処理	条件付きロジックを使用します。 変数の値を保持します。
オブザベーションの書き出し	SAS データセットまたは外部ファイルに書き出します。 出力結果をいつデータセットに書き出すかを制御します。 複数のファイルに書き出します。

詳細については、*SAS Statements: Reference* にある各ステートメントの説明を参照してください。

関数を使用したデフォルトの実行順序の変更

データの読み込みと操作は、SAS 関数を使用して実行することもできます。ステートメントや関数を使用してデータを処理する方法については、“Using Functions to Manipulate Files” (*SAS Functions and CALL Routines: Reference*)を参照してください。SAS 関数の詳細については、“SAS Functions and CALL Routines by Category” (*SAS Functions and CALL Routines: Reference*)の SAS 入出力ファイルと外部ファイルの説明を参照してください。

指定オブザベーションのフローの変更

SAS ステートメント、ステートメントオプション、データセットオプションを使用すると、特定のオブザベーションを処理する方法を変更できます。次の表は、SAS 言語要素が処理フローに及ぼす効果の一覧です。

表 18.3 プログラムの処理フローを変更する SAS 言語要素

SAS 言語要素	機能
サブセット IF ステートメント	条件が偽になった場合に現在の反復を停止します。現在のオブザベーションをデータセットに書き出さずに、DATA ステップの先頭に制御を戻します。
IF-THEN/ELSE ステートメント	現在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た後、次のステートメントへと移ります。
DO ループ	DATA ステップの一部分を複数回実行します。
LINK ステートメントと RETURN ステートメント	制御フローを変更し、指定されたラベル以降にあるステートメントを実行します。プログラムの制御を LINK ステートメントの次にあるステートメントに戻します。
FILE ステートメントの HEADER=オプション	PUT ステートメントによって新しい出力ページが開始された場合に、制御フローを変更します。RETURN ステートメントが検出されるまで、HEADER=オプションに指定されたラベル以降にあるステートメントが実行されます。RETURN ステートメントが検出されると、HEADER=オプションがアクティブになった位置に制御が戻されます。
GO TO ステートメント	処理フローを、GO TO ステートメントで指定されたラベルへと分岐します。後続のステートメントが実行された後は、DATA ステップの先頭に制御が戻されます。
INFILE ステートメントの EOF=オプション	入力ファイルの終端に達したら、実行フローを変更します。EOF=オプションに指定されたラベル以降にあるステートメントが実行されます。
IF-THEN 条件式内の自動変数_N_	DATA ステップの一部を特定の回数だけ反復します。
SELECT ステートメント	条件を満たしている場合に、ひとまとまりの SAS ステートメントを実行します。
IF-THEN 条件式内の OUTPUT ステートメント	DATA ステップの最後に達する前に、条件を満たしている場合はオブザベーションを出力します。DATA ステップの最後に結果の自動出力が実行されないようにします。
IF-THEN 条件式内の DELETE ステートメント	条件を満たしている場合に、オブザベーションを削除し、DATA ステップの先頭に処理を戻します。

SAS 言語要素	機能
IF-THEN 条件式の ABORT ステートメント	DATA ステップの実行を停止し、次の 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。ABORT ステートメントに指定するオプションや、操作方法によっては、SAS プログラムの実行を全面的に停止することもできます。
WHERE ステートメントまたは WHERE= データセットオプション	指定された 1 つ以上の条件に基づいて、特定のオブザベーションを読み込みます。

ステップ境界: ステートメントの有効時を知る

SAS プログラムステートメントが有効になる範囲(スコープ)は、ステップの終りを認識する箇所(ステップ境界)によって決定されます。そのため、ステップ境界について理解しておくことは重要です。SAS System がプログラムステートメントを実行するのは、デフォルトのステップ境界または明示的に指定されたステップ境界を越えるときだけです。たとえば、次のような DATA ステップがあるとします。

```
data _null_;  1
   set allscores(drop=score5-score7);
   title 'Student Test Scores';  2

data employees;  3
   set employee_list;
run;
```

1 DATA ステートメントにより DATA ステップが開始されます。この DATA ステートメントはステップ境界にもなります。

2 この TITLE ステートメントは、最初のステップ境界より前に置かれています。そのため、どちらの DATA ステップにも作用します。この TITLE ステートメントがグローバルステートメントです。

3 この DATA ステートメントが、最初の DATA ステップのデフォルトのステップ境界になります。

上記の例にある TITLE ステートメントは、最初のステップ境界より前に置かれています。そのため、最初の DATA ステップだけでなく 2 番目の DATA ステップにも作用します。この例では、デフォルトのステップ境界(**`data employees;`**)を使用しています。

次の例は、RUN ステートメントの後に挿入された OPTIONS ステートメントを示しています。

```
data scores;  1
   set allscores(drop=score5-score7);
run;  2

options firstobs=5 obs=55;  3

data test;
   set alltests;
run;
```

1 この DATA ステートメントは、ステップ境界になっています。

2 この RUN ステートメントは、最初の DATA ステップの明示的なステップ境界です。

3 この OPTIONS ステートメントは、2 番目の DATA ステップにのみ作用します。

この OPTIONS ステートメントでは、入力データセットから読み込む最初のオブザベーションは 5 番目、最後に読み込むオブザベーションは 55 番目と指定しています。RUN ステートメントを OPTIONS ステートメントの直前に挿入すると、SAS System が OPTIONS ステートメントを検出する前に、最初の DATA ステップがステップ境界(**run;**)に達します。そのため、OPTIONS ステートメントの設定が作用するのは、2 番目の DATA ステップだけになります。

DATA ステップの実行を開始する最も簡単な方法は、DATA ステップ内に RUN ステートメントを記述する方法です。明示的に認識される SAS ステップ境界には、次のものがあります。

- 別の DATA ステートメント
- PROC ステートメント
- RUN ステートメント

  注: SAS プログラムを対話型モードで実行する場合は、サブミットする最後のステップのステップ境界を、RUN ステートメントを使用して明示的に示す必要があります。
- セミコロン(;)(DATALINES ステートメントまたは CARDS ステートメントの場合)、またはセミコロン 4 つ(;;;;)(DATALINES4 ステートメントまたは CARDS4 ステートメントの場合)
- ENDSAS ステートメント
- SAS プログラムステートメントを含むプログラムファイルの終端(非対話型モードまたはバッチモードの場合)
- QUIT ステートメント(一部のプロシジャのみ)

対話処理中に DATA ステップをサブミットしても、SAS System がステップ境界を検出するまでは、その DATA ステップは実行されません。したがって、ステートメントの記述中にサブミットできますが、同時にステップ境界までのすべてのステートメントを入力するまでその DATA ステップは実行されません。

DATA ステップが実行を停止する原因

DATA ステップは、入力ソースの種類や数に応じて、さまざまな状況で実行を停止します。

表 18.4 *DATA ステップが実行を停止する原因*

データの読み込み	データソース	SAS ステートメント	DATA ステップが停止する条件
データなし			1 度だけ実行された後
任意のデータ			STOP ステートメントまたは ABORT ステートメントの実行時 データがすべて読み込まれたとき

データの読み込み	データソース	SAS ステートメント	DATA ステップが停止する条件
生データ	入力ストリームデータ行	INPUT ステートメント	最後のデータ行が読み込まれた後
	1 つの外部ファイル	INPUT および INFILE ステートメント	ファイル終端文字に達したとき
	複数の外部ファイル	INPUT および INFILE ステートメント	いずれかのファイルでファイル終端文字に達したとき
オブザベーションの順次読み込み	1 つの SAS データセット	SET または MODIFY ステートメント	最後のオブザベーションが読み込まれた後
	複数の SAS データセット	1 つの SET、MERGE、MODIFY、または UPDATE ステートメント	すべての入力データセットが読み込まれた後
	複数の SAS データセット	複数の SET、MERGE、MODIFY、または UPDATE ステートメント	いずれかのデータ読み込みステートメントの処理中にファイル終端文字に達したとき

DATA ステップを使用して SAS データセットからオブザベーションを読み込む場合、SET ステートメントに POINT=オプションを指定しているならば、入力用データセットの終端を検出できません。(この方法は、直接アクセスまたはランダムアクセスと呼ばれます。)このような DATA ステップでは STOP ステートメントが必要になります。

DATA ステップは、STOP ステートメントまたは ABORT ステートメントが実行されたときにも停止します。一部のシステムオプションや OBS=データセットオプションなどを使用すると、DATA ステップはそれらのオプションを使用しない場合よりも早く停止します。

VARINITCHK=システムオプションが ERROR に設定されている場合、DATA ステップは処理を停止し、変数が初期化されていなければ SAS ログにエラーを書き出します。詳細については、“VARINITCHK= System Option” (*SAS System Options: Reference*)を参照してください。

DATA ステップを用いた SAS データセットの作成について

SAS データファイルまたは SAS ビューの作成

DATA ステップでは、SAS データファイルまたは SAS ビューを作成できます。SAS データファイルは、実際のデータを保持するデータセットです。SAS ビューは、別の場所に格納されたデータを参照するデータセットです。デフォルトでは、SAS データファイルを作成します。SAS ビューを作成するには、DATA ステートメントの VIEW=オプション

を使用します。SAS ビューを使用すると、既存の DATA ステップを編集せずに現在の入力データ値を処理できます。たとえば、既存の DATA ステップを編集せずに、月次売上データを処理できます。また、出力する必要がある場合は、SAS ビューからもデータファイルを出力することができます。SAS ビューからの出力は、最新の入力データ値を動的に反映したものになります。

次の DATA ステートメントは、SAS ビュー Monthly_Sales を作成します。

```
data monthly_sales / view=monthly_sales;
```

次の DATA ステートメントは、データファイル Test_Results を作成します。

```
data test_results;
```

入力データのソース

入力データの読み込み元(入力データソース)に応じて、データを読み込む SAS ステートメントを選択する必要があります。入力データソースには、少なくとも次の 6 種類があります。

- 外部ファイル内の生データ
- ジョブストリーム内の生データ(入力ストリームデータ)
- SAS データセット内のデータ
- プログラムステートメントで作成されたデータ
- SAS カタログエントリ、クリップボード、データ URL、電子メール、FTP プロトコル、Hadoop 分散ファイルシステム、TCP/IP ソケット、URL、WebDAV プロトコル、zlib サービスのいずれかによって、リモートからアクセス可能なデータ
- データベース管理システム(DBMS)またはサードパーティベンダのデータファイルに格納されているデータ

通常、DATA ステップで読み込む入力データレコードは、上記の最初の 3 つの入力データソースのいずれかです。これらの一部または全部を組み合わせて使用することもできます。

生データの読み込み:例

例 1:外部ファイルデータの読み込み

ここでは、外部ファイルに格納されている生データを読み込んで、SAS データセットを作成する、DATA ステップの例を示します。

```
data Weight;  1
   infile 'your-input-file';  2
   input IDnumber $ week1 week16;  3
   WeightLoss=week1-week16;  4
run;  5

proc print data=Weight;  6
run;  7
```

1 DATA ステップを開始し、SAS データセット Weight を作成します。

2 データが格納されている外部ファイルを指定します。

3 レコードを読み込み、3 つの変数に値を割り当てます。

4 変数 WeightLoss の値を計算します。

5 DATA ステップを実行します。

6 PRINT プロシジャを使用してデータセット Weight を出力します。

7 PRINT プロシジャを実行します。

例 2:入力ストリームデータ行の読み込み

次の例では、入力ストリームデータ行から生データを読み込みます。

```
data Weight2;  1
   input IDnumber $ week1 week16;  2
   AverageLoss=week1-week16;  3
   datalines;  4
2477 195 163
2431 220 198
2456 173 155
2412 135 116
;  5
proc print data=Weight2;  6
run;
```

1 DATA ステップを開始し、SAS データセット Weight2 を作成します。

2 データ行を読み込み、3 つの変数に値を割り当てます。

3 変数 WeightLoss2 の値を計算します。

4 データ行の始まりです。

5 データ行の終わりをセミコロン(;)で示し、DATA ステップを実行します。

6 PRINT プロシジャを使用してデータセット Weight2 を出力します。

7 PRINT プロシジャを実行します。

例 3:欠損値を含む入力ストリームデータ行の読み込み

入力ストリームデータ行を読み込む場合、INFILE ステートメントで各種のオプションを利用できます。ここでは、MISSOVER ステートメントオプションの使用例を示します。変数に割り当てるための値を保持していないレコードがある場合は、そのレコードの変数に欠損値を割り当てます。

```
data
weight2;
   infile datalines missover;  1
   input IDnumber $ Week1 Week16;
   WeightLoss2=Week1-Week16;
   datalines;  2
2477 195  163
2431
2456 173  155
2412 135  116
;  3

proc print data=weight2;  4
run;  5
```

1 MISSOVER オプションを使用して、現在の INPUT ステートメントを満たさないレコードで、値を持たない変数に欠損値を割り当てます。

2 データ行を開始します。

3 データ行の終端を示し、DATA ステップを実行します。

4 PRINT プロシジャを使用してデータセット Weight2 を出力します。

5 PRINT プロシジャを実行します。

例 4:複数の入力ファイルを入力ストリームデータとして使用する

次の例では、複数の入力ファイルを入力ストリームデータとして読み込む方法を示します。各ファイルに含まれているレコードを読み込み、SAS データセット All_Errors を作成します。オブザベーションを変数 Station で並べ替え、並べ替えたデータセット Sorted_Errors を作成します。PRINT プロシジャを使用して、結果を出力します。

```
data all_errors;
   length filelocation $ 60;
   input filelocation;  /* reads instream data */
   infile daily filevar=filelocation
               filename=daily end=done;
   do while (not done);
      input Station $ Shift $ Employee $ NumberOfFlaws;
      output;
   end;
   put 'Finished reading ' daily=;
   datalines;
pathmyfile_A
pathmyfile_B
pathmyfile_C
;

proc sort data=all_errors out=sorted_errors;
   by Station;
run;

proc print data = sorted_errors;
   title 'Flaws Report sorted by Station';
run;
```

アウトプット 18.2 複数の入力ファイルを入力ストリームデータとして使用

Flaws Report sorted by Station

Obs	Station	Shift	Employee	NumberOfFlaws
1	Amherst	2	Lynne	0
2	Goshen	2	Seth	4
3	Hadley	2	Jon	3
4	Holyoke	1	Walter	0
5	Holyoke	1	Barb	3
6	Orange	2	Carol	5
7	Otis	1	Kay	0
8	Pelham	2	Mike	4
9	Stanford	1	Sam	1
10	Suffield	2	Lisa	1

SAS データセットからのデータの読み込み

次の例では、既存の SAS データセットからデータを読み込み、新しい変数の値を生成して、新しく作成したデータセットに格納します。

```
data average_loss;  1
   set weight;  2
   Percent=round((AverageLoss * 100) / Week1);  3
run;  4
```

1 DATA ステップを開始し、SAS データセット Average_Loss を作成します。

2 SAS データセット Weight からオブザベーションを読み込みます。

3 変数 Percent の値を計算します。

4 DATA ステップを実行します。

プログラミングステートメントを使用したデータの生成

SAS データセットのデータは、データの読み込みによってではなく、プログラムステートメントを使用してオブザベーションを生成することによっても作成できます。次の例では、入力データを読み込まない DATA ステップが、1 回だけ反復されます。

```
data investment;  1
   begin='01JAN1990'd;
   end='31DEC2009'd;
   do year=year(begin) to year(end);  2
      Capital+2000 + .07*(Capital+2000);
      output;  3
   end;
   put 'The number of DATA step iterations is '_n_;  4
```

```
run;  5

proc print data=investment;  6
   format Capital dollar12.2;  7
run;  8
```

1 DATA ステップを開始し、SAS データセット Investment を作成します。

2 1990 年から 2009 年まで毎年 2,000 ドルの資本投資を行って 7%の年利を見込む値を計算します。DO ループを 1 回反復するたびに、1 つのオブザベーションの変数値を計算します。

3 データセット Investment に各オブザベーションを書き出します。

4 DATA ステップが反復した数を示すメッセージを SAS ログに 1 回のみ書き出します。

5 DATA ステップを実行します。

6 出力を確認するには、PRINT プロシジャを使用してデータセット Investment を出力します。

7 FORMAT ステートメントを使用して、ドル記号($)、カンマ(,)、小数点(.)を含む数値を書き出します。

8 PRINT プロシジャを実行します。

DATA ステップを用いたレポートの作成

例 1: データセットを作成せずにレポートを作成する

DATA ステートメントでキーワード_NULL_をデータセット名に指定すると、データセットを作成せずにレポートを生成できます。この方法ではデータセットを作成しないため、システムリソースを節約できます。レポートには、TITLE ステートメントと FOOTNOTE ステートメントを使用して、タイトルとフットノートを配置できます。FOOTNOTE ステートメントを使用する場合は、DATA ステップの FILE ステートメントの FOOTNOTE オプションを指定してください。

```
title1 'Budget Report';  1
title2 'Mid-Year Totals by Department';
footnote 'compiled by Manager,
Documentation Development Department';  2

data _null_;  3
   set budget;  4
   file print footnote;  5
   MidYearTotal=Jan+Feb+Mar+Apr+May+Jun;  6
   if _n_=1 then  7
      do;
         put @5 'Department' @30 'Mid-Year Total';
      end;
   put @7 Department @35 MidYearTotal;  8
run;  9
```

1 タイトルを定義します。

2 フットノートを定義します。

3 DATA ステップを開始します。キーワード_NULL_によって、データセットを作成しないことを指定します。

4 データセット Budget から、1 回の反復で 1 つずつオブザベーションを読み込みます。

5 PUT ステートメントの出力結果として、PRINT ファイル参照名を使用します。デフォルトでは、PRINT ファイル参照名は、キャリッジコントロール文字とタイトルを含めることを指定します。FOOTNOTE オプションは、出力結果の各ページにフットノートを含めることを指定します。

6 反復のたびに、変数 MidYearTotal の値を計算します。

7 初回の反復に限り、レポートの変数名の見出しを書き出します。

8 反復のたびに、変数 Department と変数 MidYearTotal の現在の値を書き出します。

9 DATA ステップを実行します。

前述の例では、PRINT ファイル参照名を指定した FILE ステートメントを使用して、リスト出力を作成しています。ファイルに出力したい場合は、ファイル参照名または完全なファイル名を指定します。ファイルにキャリッジコントロール文字とタイトルを含めたい場合は、PRINT オプションを使用します。次の例は、FILE ステートメントを使用してファイル出力する方法を示しています。

```
file 'external-file' footnote print;
```

data _null_;ステートメントを使用すれば、外部ファイルに書き出すこともできます。外部ファイルへの書き出しの詳細については、*SAS Statements: Reference* にある FILE ステートメントの説明、および使用している動作環境に対応する SAS ドキュメントを参照してください。

例 2:カスタマイズレポートの作成

DATA ステップで PUT ステートメントを使用すると、カスタマイズしたレポートを作成できます。3 つの異なるセクション(ヘッダー、テーブル、フッター)を含むカスタマイズレポートの例を下記に示します。このレポートには、既存の SAS 変数の値、定数テキスト、レポート作成時に計算される値が出力されています。

アウトプット 18.3 カスタマイズレポートの例

```
Around The World Retailers EMPLOYEE BUSINESS, TRAVEL, AND TRAINING EXPENSE REPORT Employee Name:ALEJANDRO MARTINEZ Destination:CARY,
NC Departure Date:11JUL2010 Department:SALES & MARKETING Purpose of Trip/Activity:MARKETING TRAINING Return Date:16JUL2010 Trip ID#:
93-0002519                                                                                    Activity from:12JUL1993 to:16JUL2010
+-----------------------------------+--------+--------+--------+--------+--------+--------+--------+--------+
|                                   |   SUN  |   MON  |   TUE  |   WED  |   THU  |   FRI  |   SAT  |        |   PAID BY  PAID BY |
EXPENSE DETAIL                      |  07/11 |  07/12 |  07/13 |  07/14 |  07/15 |  07/16 |  07/17 | TOTALS |   COMPANY EMPLOYEE
|-----------------------------------|--------|--------|--------|--------|--------|--------|--------|--------| |Lodging,
Hotel                       |   92.96|   92.96|   92.96|   92.96|   92.96|        |        |  464.80|    464.80 |
Telephone                           |    4.57|    4.73|        |        |        |        |        |    9.30|              9.30 |
Personal Auto   36 miles @.28/mile |    5.04|        |        |        |        |    5.04|        |   10.08|             10.08 |Car
Rental, Taxi, Parking, Tolls   |        |   35.32|   35.32|   35.32|   35.32|   35.32|        |  176.60|    176.60 |Airlines, Bus,
Train (Attach Stub) |  485.00|        |        |        |        |  485.00|        |  970.00|    970.00 |
Dues                                |        |        |        |        |        |        |        |        | |Registration
Fees                    |   75.00|        |        |        |        |        |        |   75.00|             75.00 |Other (explain
below)              |        |        |        |        |        |    5.00|        |    5.00|             5.00 |Tips (excluding
meal tips)          |    3.00|        |        |        |        |    3.00|        |    6.00|            6.00
|-----------------------------------|--------|--------|--------|--------|--------|--------|--------|--------| |
Meals                               |        |        |        |        |        |        |        |        | |
Breakfast                           |        |        |        |        |        |    7.79|        |    7.79|              7.79 |
Lunch                               |        |        |        |        |        |        |        |        | |
Dinner                              |   36.00|   28.63|   36.00|   36.00|   30.00|        |        |  166.63|            166.63 |
Business Entertainment              |        |        |        |        |        |        |        |        |
|-----------------------------------|--------|--------|--------|--------|--------|--------|--------|--------| |TOTAL
EXPENSES                      |  641.57|  176.64|  179.28|  179.28|  173.28|  541.15|        | 1891.20|   1611.40   279.80
+-----------------------------------+--------+--------+--------+--------+--------+--------+--------+--------+ Travel Advance to
Employee .....................................................................................              $0.00 Reimbursement due
Employee (or ATWR) ...........................................................................            $279.80 Other:(i.e.
miscellaneous expenses and/or names of employees sharing receipt.)CAR RENTAL INCLUDE $5.00 FOR GAS APPROVED FOR PAYMENT
BY:Authorizing Manager:_____________________________________________   Emp.#  _______ Employee
Signature:_____________________________________________   Emp.#  1118 Charge to Division:ATW Region:TX Dept:MKT Acct:      6003
Date:27JUL2010
```

レポート例を生成するコードを次に示します。ユーザーは自分独自の入力データを作成する必要があります。レポート例を生成するコードについての完全な説明は、本書の範囲を超えています。次の例に関する完全な説明は、*SAS Guide to Report Writing:Examples* を参照してください。

```
options ls=132 ps=66 pageno=1 nodate;

data travel;

  /* infile 'SAS-data-set' missover; */
  infile 'c15expense.dat' missover;
 input acct div $ region $ deptchg $ rptdate : date9.
     other1-other10 /
     empid empname & $char35. / dept & $char35. /
     purpose & $char35. / dest & $char35. / tripid & $char35. /
     actdate2 : date9. /
     misc1 & $char75. / misc2 & $char75. / misc3 & $char75. /
     misc4 & $char75. /
     misc5 & $char75. / misc6 & $char75. / misc7 & $char75. /
     misc8 & $char75. /
     dptdate : date9. rtrndate : date9. automile permile /
     hotel1-hotel10 /
     phone1-phone10 / peraut1-peraut10 / carrnt1-carrnt10 /
     airlin1-airlin10 / dues1-dues10 / regfee1-regfee10 /
     tips1-tips10 / meals1-meals10 / bkfst1-bkfst10 /
```

```
     lunch1-lunch10 / dinner1-dinner10 / busent1-busent10 /
     total1-total10 / empadv reimburs actdate1 : date9.;
run;

proc format;
   value category 1='Lodging, Hotel'
                  2='Telephone'
                  3='Personal Auto'
                  4='Car Rental, Taxi, Parking, Tolls'
                  5='Airlines, Bus, Train (Attach Stub)'
                  6='Dues'
                  7='Registration Fees'
                  8='Other (explain below)'
                  9='Tips (excluding meal tips)'
                 10='Meals'
                 11='Breakfast'
                 12='Lunch'
                 13='Dinner'
                 14='Business Entertainment'
                 15='TOTAL EXPENSES';
value blanks  0=' '
            other=(|8.2|);
   value $cuscore ' '='________';
   value nuscore   . ='________';
run;

data _null_;
   file print;
   title 'Expense Report';
   format rptdate actdate1 actdate2 dptdate rtrndate date9.;
   set travel;

   array expenses{15,10} hotel1-hotel10  phone1-phone10
                         peraut1-peraut10 carrnt1-carrnt10
                         airlin1-airlin10 dues1-dues10
                         regfee1-regfee10 other1-other10
                         tips1-tips10 meals1-meals10
                         bkfst1-bkfst10 lunch1-lunch10
                         dinner1-dinner10 busent1-busent10
                         total1-total10;
   array misc{8} $ misc1-misc8;
   array mday{7} mday1-mday7;
   dptday=weekday(dptdate);
   mday{dptday}=dptdate;
   if dptday>1 then
      do dayofwk=1 to (dptday-1);
        mday{dayofwk}=dptdate-(dptday-dayofwk);
      end;
   if dptday<7 then
      do dayofwk=(dptday+1) to 7;
         mday{dayofwk}=dptdate+(dayofwk-dptday);
      end;
   if rptdate=. then rptdate="&sysdate9"d;

tripnum=substr(tripid,4,2)||'-'||substr(scan(tripid,1),6);
```

```
      put // @1 'Around The World Retailers' //

            @1 'EMPLOYEE BUSINESS, TRAVEL, AND TRAINING EXPENSE REPORT' ///

            @1 'Employee Name: ' @16 empname
            @44 'Destination: ' @57 dest
            @106 'Departure Date:' @122 dptdate /

            @4 'Department: ' @16 dept
            @44 'Purpose of Trip/Activity: ' @70 purpose
            @109 'Return Date:' @122 rtrndate /

            @6  'Trip ID#: ' @16 tripnum
            @107 'Activity from:' @122 actdate1 /

            @118 'to:' @122 actdate2 //
            @1 '+----------------------------------+--------+--------+'
               '--------+--------+--------+--------+--------+--------+' /

            @1 '|                                  |   SUN  |   MON  |'
               '   TUE  |   WED  |   THU  |   FRI  |   SAT  |        |'
               '   PAID BY  PAID BY' /

            @1 '| EXPENSE DETAIL                  '
               ' |  ' mday1 mmddyy5. ' |  ' mday2  mmddyy5.
               ' |  ' mday3 mmddyy5. ' |  ' mday4  mmddyy5.
               ' |  ' mday5 mmddyy5. ' |  ' mday6  mmddyy5.
               ' |  ' mday7 mmddyy5.
            @100 '| TOTALS |   COMPANY EMPLOYEE' ;
      do i=1 to 15;

         if i=1 or i=10 or i=15 then
           put @1 '|----------------------------------|--------|--------|'
                  '--------|--------|--------|--------|--------|--------|';
         if i=3 then
            put @1 '|' i category. @16 automile 4.0 @21 'miles @'
              @28 permile 3.2 @31 '/mile'  @37 '|' @;
            else put @1 '|' i category.  @37 '|' @;
        col=38;
        do j=1 to 10;
          if j<9 then put @col expenses{i,j} blanks8. '|' @;
             else if j=9 then put @col expenses{i,j} blanks8. @;
             else put @col expenses{i,j} blanks8.;
          col+9;
          if j=8 then col+2;
        end;
      end;
      Put @1 '+----------------------------------+--------+--------+'
             '--------+--------+--------+--------+--------+--------+' //

   @1 'Travel Advance to Employee ..............................'
             '................................................'
          @121 empadv dollar8.2 //

          @1 'Reimbursement due Employee (or ATWR) .....................'
             '................................................'
```

```
      @121 reimburs dollar8.2 //

      @1 'Other: (i.e. miscellaneous expenses and/or names of '
         'employees sharing receipt.)' /;
   do j=1 to 8;
     put @1 misc{j} ;
   end;
   put / @1  'APPROVED FOR PAYMENT BY: Authorizing Manager:'
      @48 '_____________________________________________'
      @100 'Emp. #  _______' ///

      @27 'Employee Signature:'
      @48 '_____________________________________________'
      @100 'Emp. #  ' empid ///

      @6 'Charge to Division:' @26 div $cuscore.
      @39 'Region:'            @48 region $cuscore.
      @59 'Dept:'              @66 deptchg $cuscore.
      @79 'Acct:'              @86 acct nuscore.
      @100 'Date:'             @107 rptdate /
      _page_;
run;
```

例 3:ODS と DATA ステップを使用した HTML レポートの作成

```
ods html body='your_file.html';

title 'Leading Grain Producers';
title2 'for 2012';

proc format;
   value $cntry 'BRZ'='Brazil'
                'CHN'='China'
                'IND'='India'
                'INS'='Indonesia'
                'USA'=''United States';
run;

data _null_;
   length Country $ 3 Type $ 5;
   input Year country $ type $ Kilotons;
   format country $cntry.;
   label type='Grain';

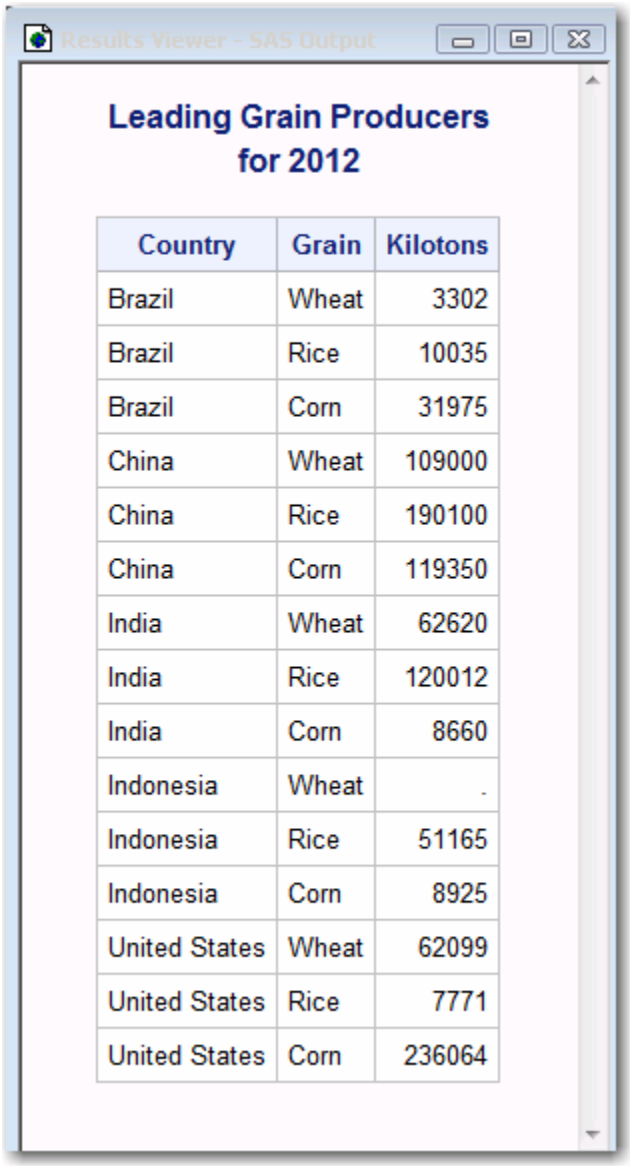
file print
ods=(variables=(country type kilotons));

put _ods_;

   datalines;
2012 BRZ  Wheat    3302
2012 BRZ  Rice     10035
2012 BRZ  Corn     31975
```

```
2012 CHN  Wheat    109000
2012 CHN  Rice     190100
2012 CHN  Corn     119350
2012 IND  Wheat    62620
2012 IND  Rice     120012
2012 IND  Corn     8660
2012 INS  Wheat    .
2012 INS  Rice     51165
2012 INS  Corn     8925
2012 USA  Wheat    62099
2012 USA  Rice     7771
2012 USA  Corn     236064
;
run;
```

アウトプット 18.4 *ODS により生成される HTML ファイル*



Leading Grain Producers for 2012

Country	Grain	Kilotons
Brazil	Wheat	3302
Brazil	Rice	10035
Brazil	Corn	31975
China	Wheat	109000
China	Rice	190100
China	Corn	119350
India	Wheat	62620
India	Rice	120012
India	Corn	8660
Indonesia	Wheat	.
Indonesia	Rice	51165
Indonesia	Corn	8925
United States	Wheat	62099
United States	Rice	7771
United States	Corn	236064

DATA ステップと ODS

ODS (Output Delivery System)は、さまざまな形式で出力を配信し、その形式へのアクセスを簡単にする方法です。ODS は、DATA ステップまたは PROC ステップから出力構造を定義するテンプレートを使用します。DATA ステップでは、FILE ステートメントと PUT ステートメントにより、ODS オプションを使用できます。

ODS は、生データを 1 つ以上のテンプレートと結合することにより、出力オブジェクトと呼ばれる各種の出力を作成します。出力オブジェクトは、LISTING 出力先、PRINTER 出力先、HTML 出力先などの出力先へと送信されます。詳細については、“SAS 出力先の指定とカスタマイズ” (147 ページ)を参照してください。ODS に関する詳細については、*SAS Output Delivery System: User's Guide* を参照してください。

DATA ステップ処理時間

DATA ステップ処理時間には 2 つのステージがあります。1 つ目は起動(またはコンパイル時間)、2 つ目は実行時間です。コンパイル時間は、SAS コンパイラで SAS ソースコードがスキャンされ、実行可能プログラムに変換されるのにかかる時間です。実行時間は、SAS が SAS ファイル内の各オブザベーションに対して DATA ステップを実行するのにかかる時間です。この 2 つのフェーズが同時に発生することはありません。つまり、DATA ステップは、まずコンパイルしてから実行します。これら 2 つのフェーズの詳細については、“コンパイルフェーズ” (392 ページ)および“実行フェーズ” (392 ページ)を参照してください。

これらの処理時間、および SAS プログラム構造との関係を理解しておくと、パフォーマンス向上の方法を探す際に役立つことがあります。一般に、DATA ステップが処理するステートメントが多いほど、コンパイル時間が長くなります。これに対して、多数のオブザベーションを処理する DATA ステップは、入出力操作の回数が多くなるため、実行時間が長くなる傾向があります。

たとえば、入出力操作の回数が少ない(すなわち、処理するオブザベーション数が比較的少ない)非常に大きな DATA ステップジョブは、複雑さを軽減したり、繰り返しや使用しないコードを排除したりするために書き換えが必要な場合もあります。DO ループや、PROC FCMP で作成されるユーザー定義関数は、コンパイルを要するコードの量を減らしてコンパイル時間を削減するために使用可能なメソッドです。CPU を集中的に使用するプログラムを実行する際のパフォーマンス向上の詳細については、“CPU パフォーマンスを最適化する手法” (192 ページ)を参照してください。

DATA ステップで、数百のオブザベーションを処理することにほとんどの時間が費やされている場合は、I/O を最適化するための別のテクニックの方が役に立つこともあります。入出力操作の回数が多いプログラムを実行する際のパフォーマンス向上の詳細については、“I/O 最適化の手法” (185 ページ)を参照してください。

いくつかの SAS システムオプションでは、処理時間を最小限に抑え、パフォーマンスを最適化するのに役立つ情報が提供されます。たとえば、SAS の FULLSTIMER オプションは、各 DATA ステップのパフォーマンス統計量を収集して表示するので、データ処理の各ステップでどのリソースが使用されたかを判別できます。このオプションと一般的な最適化の詳細については、12 章, “システムパフォーマンスの最適化” (183 ページ)を参照してください。

次の例は、少数のオブザベーションを有する非常に大きな DATA ステップジョブのコンパイル時間推定方法を示しています。このプログラムは、DATETIME 関数を%PUT マクロステートメントと一緒に使用して、コンパイル開始時間を計算します。次に、_N_

自動変数を使用して、実行開始時間を求めます(この変数は、実行フェーズ開始時に SAS が常に 1 に設定します)。2 つの時間の差を計算することにより、プログラムは DATA ステップの合計コンパイル時間を戻します。

例のコード 18.1 *Finding Compilation and Execution Time*

```
options nosource;
%put Starting compilation of DATA step: %QSYSFUNC(DATETIME(), DATETIME20.3);
%let startTime=%QSYSFUNC(DATETIME());

data a;
  if _N_ = 1 then do;
    endTime = datetime();
    put 'Starting execution of
         DATA step: ' endTime:DATETIME20.3;
    timeDiff=endTime-&startTime;
    put 'The Compile time for this DATA Step is
      approximately ' timeDiff:time20.6;
end;
 /* Lots of DATA step code */
run;
```

アウトプット 18.5 *コンパイル時間と実行時間を求める場合のログ出力*

```
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds


Starting compilation of DATA step: 29JUN12:16:17:54.725

Starting execution of DATA step: 29JUN12:16:17:54.755
The Compile time for this DATA Step is approximately 0:00:00.030000
NOTE: The data set WORK.A has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

注: マクロステートメントとマクロ変数は、コンパイル時に解決されるので、DATA ステップの実行にかかる時間には関係ありません。マクロアクティビティでのステートメント処理については、“Getting Started with the Macro Facility” (*SAS Macro Language: Reference*)および“SAS Programs and Macro Processing” (*SAS Macro Language: Reference*)を参照してください。

19 章

生データの読み込み

生データの読み込みの定義

生データ
生データとは、SAS データセットに読み込まれていない、処理される前のデータです。DATA ステップを使用すると、次の 2 種類の入力データソースから、生データを SAS データセットに読み込むことができます。

- 入力ストリームデータ
- 外部ファイル

注: 生データには、データベース管理システム(DBMS)のファイルは含まれません。DBMS ファイルに格納されたデータを読み込むには、SAS/ACCESS ソフトウェアが必要です。SAS/ACCESS 機能の詳細については、31 章, "SAS/ACCESS ソフトウェアについて" (699 ページ)を参照してください。

生データの読み込み手順

生データを読み込むには、次に示す方法のいずれかを使用します。

- SAS ステートメント
- SAS 関数
- 外部ファイルインターフェイス(EFI)
- インポートウィザード

DATA ステップを使用して生データを読み込む場合、INPUT ステートメント、DATALINES ステートメント、INFILE ステートメントを組み合わせて使用できます。これらのステートメントを使用すると、自動的にデータが SAS データセットに読み込まれます。これらのステートメントに関する詳細については、"INPUT ステートメントを用いた生データの読み込み" (423 ページ)を参照してください。

SAS 関数を利用して、外部ファイルの操作と生データのレコードの読み込みを行うこともできます。SAS 関数では、生データをより柔軟に取り扱うことができます。使用できる関数の詳細については、"SAS Functions and CALL Routines by Category" (*SAS Functions and CALL Routines: Reference*)にある SAS 入出力ファイルと外部ファイルの説明を参照してください。ステートメントや関数によるファイルの処理の相違については、"Using Functions to Manipulate Files" (*SAS Functions and CALL Routines: Reference*)を参照してください。

動作環境がグラフィカルユーザーインターフェイスをサポートしている場合は、EFI またはインポートウィザードを使用して生データを読み込むことができます。EFI は、ポイントアンドクリックのグラフィカルインターフェイスで、SAS ソフトウェアの内部形式ではないデータを読み書きすることができます。EFI を使用すると、外部ファイルを読み込み、SAS データセットへ書き出すことができます。また、SAS データセットからデータを読み込んで、外部ファイルへ書き出すこともできます。EFI の詳細については、*SAS/ACCESS Interface to PC Files: Reference* を参照してください。

注: EFI に渡すデータファイルがパスワードで保護されている場合、ログイン ID とパスワードを何回も入力するよう求められます。

インポートウィザードは、外部データ入力データソースから読み込んだデータを、SAS データセットに書き出す操作を行います。インポートウィザードが表示する一連のウィ

ンドウで選択を行うだけで、ユーザーは必要な処理をすべて完了できます。ウィザードの詳細については *SAS/ACCESS Interface to PC Files: Reference* を参照してください。

動作環境の情報
SAS ジョブで外部ファイルを使用する場合は、使用している動作環境での表記規則に従って、適切な構文でファイル名を指定する必要があります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

データの種類

定義

データ値
文字値または数値です。

数値
数字と、必要に応じて小数点(.)あるいは負符号(-)を付けて表記します。SAS データセットに読み込まれた数値は、動作環境に固有の浮動小数点形式で格納されます。数字以外の文字が含まれた非標準の数値については、入力形式を利用することで、読み込むことができます。

文字値
1 文字以上の文字列を指定できます。

標準データ
リスト入力、カラム入力、フォーマット入力、名前付き入力を利用して読み込むことのできる文字値または数値です。標準データの例を次に示します。

- `ARKANSAS`
- `1166.42`

非標準データ
入力形式を利用した場合にのみ、読み込むことのできるデータです。非標準データの例としては、カンマ(,)、ドル記号($)、ブランクを含む数値、日時値、16 進表記の値、バイナリ値があります。

数値データ

数値データは、いくつかの形式で表現されます。標準数値データを読み込む場合、特殊な入力形式を指定する必要はありません。非標準数値データを読み込むには、入力形式を指定する必要があります。表 19.2 (420 ページ)は、標準、非標準、無効な数値データ値と、それらを読み込む方法が示されています。すべての SAS 入力形式に関する完全な説明については、*SAS Formats and Informats: Reference* を参照してください。

表 19.1 *標準数値データの読み込み*

データ	説明	解法
`23`	右揃えの数値	なし
`23`	桁揃えのない数値	なし

データ	説明	解法
23	左揃えの数値	なし
00023	前置ゼロのある数値	なし
23.0	小数点のある数値	なし
2.3E1	指数表記の数値。2.30 (ss1)	なし
230E-1	指数表記の数値。230x10 (ss-1)	なし
-23	先頭に負符号(-)が付いた負の数値	なし

表 19.2 非標準数値データの読み込み

データ	説明	解法
2 3	ブランクを含む数値	COMMA.入力形式または BZ.入力形式
- 23	ブランクを含む数値	COMMA.入力形式または BZ.入力形式
2,341	カンマを含む数値	COMMA.入力形式
(23)	かっこ	COMMA.入力形式
C4A2	16 進表記の数値	HEX.入力形式
1MAR90	日付値	DATE.入力形式

表 19.3 無効数値データの読み込み

データ	説明	解法
23 -	数字の後に負符号(-)が付いた数値	負符号(-)を数字より前の先頭に付けるか、プログラムを記述して処理します。S370FZDTw.d 入力形式を使用することもできますが、正の数値の後ろに正符号(+)が必要になります。

データ	説明	解法
`..`	2 個の小数点	1 個の小数点を欠損値とするプログラムを記述するか、または INPUT ステートメント内で??修飾子を使用することにより、無効な入力値を欠損値とするプログラムを記述します。
`J23`	文字値の数字	文字値として読み込むか、生データを編集して有効な数値に変換します。

数値データの読み込みには次の規則が適用されます。

- 数字の前にブランクを空けずにかっこまたは負符号を付けると、負の数値を表します。
- 数値データの先頭に付けられたゼロは、変数に割り当てられる値には影響しません。数値データに付けられた前置ゼロと前後のブランクは、変数には格納されません。SAS System 以外の一部のプログラミング言語とは異なり、SAS System のデフォルトでは、後置ブランクは 0 としては読み込まれません。後置ブランクを 0 として読み込むには、BZ.入力形式を使用します。詳細については *SAS Formats and Informats: Reference* を参照してください。
- 前置ブランクあるいは後置ブランクの付いた数値データを、読み込むことができます。ただし、ブランクが埋め込まれた数値データは、COMMA.入力形式または BZ.入力形式を使用して読み込む必要があります。
- 明示的な小数点のない小数値を入力データ行から読み込むには、小数点の位置を指定する必要があります。カラム入力を利用した小数値の読み込み、あるいはフォーマット入力を利用した入力形式を使用します。詳細については、*SAS Formats and Informats: Reference* にある INPUT ステートメントの説明を参照してください。入力データに明示的な小数点を付けると、INPUT ステートメント内の小数点指定はすべて上書きされます。

文字データ

INPUT ステートメントを使用して読み込まれた値は、次のいずれかに当てはまる場合は文字値とみなされます。

- INPUT ステートメント内の変数名の後に、ドル記号($)が付く場合。
- 文字入力形式が使用されている場合。
- すでに変数が文字として定義されている場合。たとえば、LENGTH ステートメント、RETAIN ステートメント、割り当てステートメントによって、または式の中で変数がすでに文字として定義されている場合、その変数は文字値とみなされます。

文字変数に格納する入力データには、どのような文字が入っていてもかまいません。前置ブランクとセミコロン(;)が含まれる生データの場合は、次の表に示すガイドラインに従ってください。

表 19.4 前置ブランクとセミコロンを含んでいる入力ストリームデータまたは外部ファイルの読み込み

データ内の文字	使用する手段	理由
保存したい前置ブランクまたは後置ブランクがある場合	フォーマット入力と$CHAR*w*.入力形式を使用する。	リスト入力を使用すると、変数に文字列を割り当てる前に、文字値から前置ブランクと後置ブランクが削除されます。
入力ストリームデータにセミコロン(;)がある場合	DATALINES4 ステートメントまたは CARDS4 ステートメントで、データ終端を表す 4 個のセミコロン(;;;;)を使用する。	通常の DATALINES ステートメントと CARDS ステートメントでは、セミコロン(;)はデータ終端を表します。
区切り文字、空白文字、引用文字列がある場合	DSD オプションと、DLM=または DLMSTR=オプションのいずれかを指定した INFILE ステートメントを使用する。	これらのオプションを使用すると、引用文字列内に区切り文字が含まれている場合でも文字値を読み込めます。また、区切り文字が 2 個連続している場合に欠損値として扱うことや、文字値から引用符を削除することもできます。

文字データを読み込む場合、次の点に注意する必要があります。

- INPUT ステートメントを使用して読み込む場合、変数名の後ろにドル記号($)を付けると、入力データ行から読み込まれた文字データの大文字と小文字の違いが保持されます。入力データ行から読み込む文字データをすべて大文字として読み込む場合は、CAPS システムオプションを使用するか、または$UPCASE 入力形式を使用します。
- 変数の長さに満たない値は、指定された長さにするために、値の末尾にブランクが追加されます。この処理は、ブランクの埋め込みと呼ばれます。

生データのソース

入力ストリームデータ

次の例では、INPUT 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んでいます。

```
data weight;
   input PatientID $ Week1 Week8 Week16;
   loss=Week1-Week16;
   datalines;
2477 195 177 163
2431 220 213 198
2456 173 166 155
2412 135 125 116
```

```
;
```

注: データ行の直後にあるセミコロン(;)のみの行は、データ行の終端を表すための規則です。ただし、PROC ステートメント、DATA ステートメント、グローバルステートメントでは、最後のデータ行の直後にセミコロン(;)のみの行を記述すると、直前の DATA ステップのサブミットも行われます。

セミコロンを含む入力ストリームデータ

次の例では、セミコロンを含む入力ストリームデータを読み込んでいます。

```
data weight;
   input PatientID $ Week1 Week8 Week16;
   loss=Week1-Week16;
   datalines4;
24;77 195 177 163
24;31 220 213 198
24;56 173 166 155
24;12 135 125 116
;;;;
```

外部ファイル

次の例は、INFILE ステートメントと INPUT ステートメントを使用して、外部ファイルから生データを読み込む方法を示しています。

```
data weight;
   infile file-specification or path-name;
   input PatientID $ Week1 Week8 Week16;
   loss=Week1-Week16;
run;
```

注: INFILE ステートメントを使用してファイルを指定する方法については、使用している動作環境に対応する SAS ドキュメントを参照してください。

INPUT ステートメントを用いた生データの読み込み

入力スタイルの選択

INPUT ステートメントは、入力ストリームデータ行または外部ファイルから、生データを SAS データセットへと読み込みます。レコードにあるデータ値の記述形式に応じて、次に示す入力スタイルを使用できます。

- リスト入力
- カラム入力
- フォーマット入力
- 名前付き入力

1 つの INPUT ステートメント内で、複数の入力スタイルを組み合わせて使用することもできます。入力スタイルの詳細については、*SAS Statements: Reference* にある INPUT ステートメントの説明を参照してください。

リスト入力

リスト入力では、スキャン方式を使用してデータ値を見つけます。データ値は、複数の列に整えられていなくてもかまいませんが、1 つ以上のブランクか、定義済みの区切り文字(デリミタ)で区切られている必要があります。リスト入力では、変数名の指定が必要です。文字変数を定義する場合には、変数名の後ろにドル記号($)も付加します。データフィールドの位置を指定する必要はありません。

リスト入力の例を次に示します。

```
data scores;
   length name $ 12;
   input name $ score1 score2;

datalines;
Riley 1132 1187
Henderson 1015 1102
;
```

リスト入力でデータを読み込む場合は、次のような制約があります。

- 入力値は、1 個以上のブランク(デフォルトの区切り文字)か、INFILE ステートメントの DLM=または DLMSTR=オプションで指定された区切り文字で区切られている必要があります。連続する区切り文字を読み込む場合に、それらの間に欠損値があるものとして処理したいときは、INFILE ステートメントの DSD オプションを指定します。
- ブランクで欠損値を表現することはできません。ピリオド(.)などの値を使用する必要があります。
- 8 バイトを超える文字入力値を読み込んで格納するには、INPUT ステートメントを記述する前に、LENGTH ステートメント、INFORMAT ステートメント、ATTRIB ステートメントのいずれかを使用するか、入力形式と INPUT ステートメントでコロン(:)修飾子を使用したリスト入力を使用して、変数の長さを定義します。詳細については、“修飾リスト入力” (424 ページ) を参照してください。
- 文字値がブランクで区切られている場合、文字値の中に埋め込みブランクを含めることはできません。
- データフィールドは順番に読み込まれます。
- 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。

注: パック 10 進データなどの非標準数値データを読み込む場合は、フォーマット入力を使用する必要があります。詳細については、“フォーマット入力” (426 ページ) を参照してください。

修飾リスト入力

修飾リスト入力は、フォーマット修飾子を含んだ、より柔軟性のあるリスト入力です。次のフォーマット修飾子を指定して SAS 入力形式を使用すると、リスト入力を利用して非標準データを読み込めるようになります。

- アンパサンド(&)フォーマット修飾子を指定すると、リスト入力を使用して、1 つ以上の埋め込みブランクを持つ文字値を読み込むことや、文字入力形式の指定が行えます。SAS System は、2 個の連続するブランクを検出するか、定義されている変数の長さまたは入力行の最後に達するまで、読み込みを行います。

- コロン(:)フォーマット修飾子を指定すると、リスト入力を使用することや、変数名の後に情報(文字値であるかそれとも数値であるか)を指定することができます。SAS System は、空のカラムを検出するか、定義されている変数の長さ(文字のみ)またはデータ行の最後に達するまで、読み込みを行います。
- チルダ(~)フォーマット修飾子を指定すると、文字データにある、一重引用符(')、二重引用符(")、区切り文字を読み込み、格納することができます。

コロン(:)およびチルダ(~)フォーマット修飾子の使用例を次に示します。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。そうしない場合、INPUT ステートメントはチルダ(~)フォーマット修飾子を無視します。

```
data scores;
   infile datalines dsd;
   input Name : $9. Score1-Score3 Team ~ $25. Div $;

datalines;
Smith,12,22,46,"Green Hornets, Atlanta",AAA
Mitchel,23,19,25,"High Volts, Portland",AAA
Jones,09,17,54,"Vulcans, Las Vegas",AA
;
proc print data=scores;
```

アウトプット 19.1 *フォーマット修飾子を指定した出力結果の例*

The SAS System

Name	Score1	Score2	Score3	Team	Div
Smith	12	22	46	"Green Hornets, Atlanta"	AAA
Mitchel	23	19	25	"High Volts, Portland"	AAA
Jones	9	17	54	"Vulcans, Las Vegas"	AA

カラム入力

カラム入力を使用すると、入力データ行にある、複数の列に整えられた標準データ値を読み込むことができます。変数名を指定し、文字変数の場合はその後にドル記号($)を付加します。さらに、入力値が入力データ行のどのカラムに位置するかを指定します。

```
data scores;
   infile datalines truncover;
   input name $ 1-12 score2 17-20 score1 27-30;

datalines;
Riley           1132       987
Henderson       1015      1102
;
```

注: さまざまな長さを持つ複数のデータ値が適切に処理されるようにするには、INFILE ステートメントの TRUNCOVER オプションを使用します。

カラム入力を使用するには、データ値が次の条件を満たしている必要があります。

- すべての入力行で、同じフィールドに格納されていること。
- 標準の数値形式または文字形式のデータであること。

注: カラム入力では入力形式を使用できません。

カラム入力には、次の特徴があります。

- 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。
- 1 個のピリオド(.)など、欠損値のための記述は不要です。
- 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。
- 値の全部または一部を再読み込みできます。
- 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。
- ブランクなどの区切り文字で値を区切る必要はありません。

注意:

DATALINES ステートメントにおいてカラム形式でデータを入力すると同時にタブを挿入した場合、予期しない結果が発生することがあります。SAS 拡張エディタまたは SAS プログラムエディタを使用する場合、問題が発生します。問題を避けるには、次のいずれかを実行します。

- SAS 以外の別エディタを使用して、データのすべてのタブを単一スペースに置換します。
- SAS エディタから%INCLUDE ステートメントを使用して、コードをサブミットします。
- SAS 拡張エディタを使用している場合は、**ツール** ⇨ **オプション** ⇨ **拡張エディタ**を選択して、タブサイズを 4 から 1 に変更します。

フォーマット入力

フォーマット入力は、入力形式が持つ柔軟性と、カラム入力が持つさまざまな特徴を兼ね備えています。フォーマット入力を使用すると、SAS System に追加の命令を指定することで非標準データを読み込めます。フォーマット入力では、通常はポインタコントロールを使用します。これによって、データの読み込み時に、入力バッファ内の入力ポインタの位置を制御できます。

次の DATA ステップにある INPUT ステートメントでは、入力形式とポインタコントロールを使用しています。$12.と COMMA5.は入力形式で、+4 と+6 はカラムポインタコントロールです。

```
data scores;
   input name $12. +4 score1 comma5. +6 score2 comma5.;

datalines;
Riley           1,132      1,187
Henderson       1,015      1,102
;
```

注: 入力形式を使用すると、複数の列へと整えられていないデータを読み込むこともできます。詳細については、“修飾リスト入力” (424 ページ) を参照してください。

フォーマット入力に関する注意点を下記に示します。

- 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。

- 1 個のピリオド(.)など、欠損値のための記述は不要です。
- ポインタコントロールを使用してポインタの位置を制御すると、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。
- 値の全部または一部を再読み込みできます。
- フォーマット入力を使用することで、パック 10 進データやカンマ区切りの数字などの、非標準形式データを読み込めます。

名前付き入力

名前付き入力を使用すると、変数名と等号(=)に続いてデータ値を記述したレコードを読み込むことができます。次の INPUT ステートメントでは、等号を含むデータ行を読み込んでいます。

```
data games;
   input name=$ score1= score2=;

datalines;
name=riley score1=1132 score2=1187
;
```

注: INPUT ステートメントの変数の直後に等号(=)の記述があると、SAS System は、入力データ行の残りのデータに名前付き入力値しか格納されていないものとみなします。名前付き入力を使用した後は、同じ INPUT ステートメント内で別の入力形式へと切り替えることはできません。また、名前付き入力のデータ値に対応する変数が INPUT ステートメント中で定義されていない場合、フィールドが欠損していることを示すメッセージが SAS ログに表示されます。

その他のデータ読み込み機能

入力スタイル以外にも、SAS System にはさまざまなデータ読み込み機能が数多く用意されています。INFILE ステートメントを INPUT ステートメントと組み合わせてオプションを使用すると、データレコードの読み込みをよりきめ細かく制御できます。次の表には、一般的なデータ読み込みタスクと、それに対応する INPUT ステートメントおよび INFILE ステートメントで使用可能な機能が示されています。

表 19.5 *その他のデータ読み込み機能*

入力	目的	使用するもの
複数のレコード	1 個のオブザベーションを作成	#n 行ポインタコントロールまたは/行ポインタコントロールを使用した、DO ループを持つ INPUT ステートメント
単一のレコード	複数のオブザベーションの作成	後置@@を指定した、INPUT ステートメント 後置@を指定した、複数の INPUT ステートメントと OUTPUT ステートメント

入力	目的	使用するもの
可変長データフィールドおよび可変長レコード	区切り文字で区切られたデータ値の読み込み	INPUT ステートメント内でフォーマット修飾子(省略可)を使用したリスト入力。および TRUNCOVER オプション、DLM=オプション、DLMSTR=オプション、DSD オプションのいずれかを指定した INFILE ステートメント
	区切り文字で区切られていないデータ値の読み込み	$VARYING*w*.入力形式を使用した INPUT ステートメント、および LENGTH=オプションと TRUNCOVER オプションを使用した INFILE ステートメント
さまざまなレコードレイアウトを持つ 1 個のファイル		IF-THEN ステートメントを使用し、必要に応じて後置@または後置@@を指定した複数の INPUT ステートメント
階層構造を持つ複数のファイル		IF-THEN ステートメントを使用し、必要に応じて後置@を指定した複数の INPUT ステートメント
複数の入力ファイルを使用する場合、または EOF 到達時にプログラムフローを制御する場合		EOF=オプションまたは END=オプションを使用した INFILE ステートメント 複数の INFILE ステートメントと INPUT ステートメント FILEVAR=オプションを使用した INFILE ステートメント 連結、ワイルドカード、パイプのいずれかを使用した FILENAME ステートメント
各レコードの一部分のみ		LINESIZE=オプションを使用した INFILE ステートメント
ファイル内の一部のレコード		FIRSTOBS=オプションと OBS=オプションを使用した INFILE ステートメント、FIRSTOBS=システムオプションと OBS=システムオプション、#n 行ポインタコントロール
入力ストリームデータ行	特殊オプションを使用して読み込みを制御	DATALINES オプションと適切なオプションを使用した INFILE ステートメント

入力	目的	使用するもの
特定のカラムから読み込みを開始する場合		@カラムポインタコントロール
前置ブランク	同ブランクの維持	$CHARw.入力形式を使用した INPUT ステートメント
ブランク以外の区切り文字(コロン修飾子を 使用したリスト入力または修飾リスト入力)		DLM=オプションか DLMSTR=オプション、および DSD オプションを指定した INFILE ステートメント
標準タブ文字		DLM=オプションまたは DLMSTR=オプションを使用した INFILE ステートメント、または EXPANDTABS オプションを使用した INFILE ステートメント
欠損値(コロン修飾子を使用した リスト入力または修飾リスト入力)	データの整合性を損なわずにオブザベーションを作成 デフォルト動作を無効にしてデータの整合性を保護	TRUNCOVER オプションを指定した INFILE ステートメント。DLM=オプションまたは DLMSTR=オプション、および DSD オプションのいずれかまたは両方が必要となる場合もあります。

データ読み込み機能に関する詳細については、*SAS Statements: Reference* にある INPUT ステートメントおよび INFILE ステートメントの説明を参照してください。

SAS における無効データの取り扱い

次の場合には、入力データ値が無効になります。

- 指定した入力形式が、使用できない場合。
- 入力データの形式が、指定された入力形式に従っていない場合。
- 入力データの形式が、使用されている入力スタイルと一致していない場合。たとえば、標準数値データ(ドル記号($)や入力形式なし)として読み込んだデータが、標準 SAS 数値データの規則に従っていない場合などです。
- 入力データが、指定できる値の範囲に適合していない場合。

動作環境の情報
数値の範囲は、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

SAS System は、読み込むデータ値が指定された変数の種類と一致しない場合、その値を指定された種類に変換しようとします。変換できない場合はエラーが発生し、次の処理が実行されます。

- 読み込み中の変数の値が欠損値に設定されます。または、INVALIDDATA=システムオプションで指定された値に設定されます。

- 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。
- 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。
- 無効なデータが含まれている入力行とカラム番号が、SAS ログに出力されます。データ行に表示不能な文字が含まれている場合、その文字は 16 進表記で出力されます。カラム番号を見やすくするために、入力行の上部にスケール目盛りが出力されます。

生データ内にある欠損値の読み込み

入力データ内における欠損値の表現

読み込まれる大量のデータの中には、欠損値が含まれていることがあります。SAS System では、これらの値は欠損しているものとして認識します。生データを読み込む場合、次の文字を使用して欠損値を表現できます。

数値欠損値
: 1 個のピリオド(.)で表します。リスト入力を除くすべての入力スタイルでは、数値欠損値をブランクでも表現できます。

文字欠損値
: 1 個のブランクで表します。ただし例外として、リスト入力では欠損値を表すのに 1 個のピリオド(.)を使用する必要があります。

特殊数値欠損値
: 小数点の後に 1 つの文字またはアンダースコア(_)が続く 2 つの文字で表します。

欠損値の詳細については、5 章, “欠損値” (77 ページ)を参照してください。

数値入力データ内にある特殊な欠損値

SAS System では、数値データ内にある各種の欠損値を区別することができます。数値変数の場合は、A - Z までのアルファベット(大文字または小文字)とアンダースコア(_)を使用することで、特殊欠損値を 27 個まで表現できます。

次の例は、DATA ステップ内で、MISSING ステートメントを使用して欠損値を読み込むプログラムを示しています。

```
data test_results;
   missing a b c;
   input name $8. Answer1 Answer2 Answer3;

datalines;
Smith    2 5 9
Jones    4 b 8
Carter   a 4 7
Reed     3 5 c
;
```

式や割り当てステートメントで特殊欠損数値を指定する場合は、次のようにピリオドを使用する必要があります。

```
x=.d;
```

ただし、入力データでは、特殊欠損数値のデータ値それぞれに 1 個ずつピリオドを指定する必要はありません。たとえば、次の DATA ステップでは入力データ行内の特殊

欠損値にピリオド(.)を使用していますが、ピリオドのない入力データと同じ結果を得ることができます。

```
data test_results;
   missing a b c;
   input name $8. Answer1 Answer2 Answer3;
   datalines;
Smith    2 5 9
Jones    4 .b 8
Carter   .a 4 7
Reed     3 5 .c
;

proc print;
run;
```

アウトプット 19.2 *特殊数値欠損値を持つデータの出力結果*

The SAS System

Obs	name	Answer1	Answer2	Answer3
1	Smith	2	5	9
2	Jones	4	B	8
3	Carter	A	4	7
4	Reed	3	5	C

注: 特殊欠損値は、アルファベットの大文字で表示および出力されます。

バイナリデータの読み込み

定義

バイナリデータ
: バイナリ形式で格納されている数値データです。バイナリ数値は、基数が 2 で、算用数字の 0 と 1 を用いて表されます。

パック 10 進データ
: 1 バイトを使用して 10 進数の 2 桁を表すようにエンコードされた、バイナリ 10 進数です。パック 10 進表現では、10 進数データを正確な精度で格納します。仮数と指数が分かれていないため、数値の小数部は入力形式または出力形式を使用して決定する必要があります。

ゾーン 10 進データ
: 1 バイトを使用して 1 桁を格納するようにエンコードされた、バイナリ 10 進数です。最後のバイトには、最後の桁だけでなく、数値の正負符号も格納されます。ゾーン 10 進数は表示可能な表現を作成します。

バイナリ入力形式の使用

バイナリ形式のデータを読み込むには、SAS 入力形式による特殊な命令を使用します。INPUT ステートメント内では、フォーマット入力を使用したり、入力形式を指定したりすることもできます。バイナリデータを読み込む場合に指定する SAS 入力形式は、次の要因によって決定します。

- 読み込む数値データのタイプ(バイナリデータ、パック 10 進データ、ゾーン 10 進データ、またはそれらの組み合わせ)
- データが作成されたシステムのタイプ
- データの読み込みに使用するシステムのタイプ

バイナリ数値データの格納形式は、コンピュータプラットフォームによって異なります。バイトの順序(ビッグエンディアンまたはリトルエンディアン)も、コンピュータプラットフォームによって異なります。詳細については、“Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” (*SAS Formats and Informats: Reference*)を参照してください。

SAS System は、バイナリデータの読み込み用の入力形式と、バイナリデータの書き出し用の出力形式を数多く提供しています。これらの入力形式のうち、いくつかはプラットフォームに固有のネイティブモードでデータを読み込みます。つまり、SAS System が稼働しているコンピュータでの標準のバイト順序システムを使用してデータを読み込みます。ほかの入力形式では、ネイティブモードとは無関係に、データは強制的に IBM 370 モードで読み込みます。次の表は、データをネイティブモードまたは IBM 370 モードで読み込む入力形式の対応を示す一覧です。

表 19.6 *ネイティブモードの入力形式と IBM 370 モードの入力形式*

説明	ネイティブモードの入力形式	IBM 370 モードの入力形式
ASCII 文字	$*w.*	$ASCII*w.*
ASCII 数値	*w.d*	$ASCII*w.*
EBCDIC 文字	$*w.*	$EBCDIC*w.*
EBCDIC 数値(標準)	*w.d*	S370FF*w.d*
バイナリ整数	IB*w.d*	S370FIB*w.d*
正のバイナリ整数	PIB*w.d*	S370FPIB*w.d*
バイナリ実数	RB**w.d**	S370FRB*w.d*
符号なしバイナリ整数	PIB*w.d*	S370FIBU*w.d*、S370FPIB*w.d*
パック 10 進データ	PD*w.d*	S370FPD*w.d*
符号なしパック 10 進データ	PK*w.d*	S370FPDU*w.d* または PK*w.d*

説明	ネイティブモードの入力形式	IBM 370 モードの入力形式
ゾーン 10 進データ	ZD*w.d*	S370FZD*w.d*
前符号付きゾーン 10 進データ	S370FZDL*w.d*	S370FZDL*w.d*
分離した前符号付きゾーン 10 進データ	S370FZDS*w.d*	S370FZDS*w.d*
分離した後符号付きゾーン 10 進データ	S370FZDT*w.d*	S370FZDT*w.d*
符号なしゾーン 10 進データ	ZD*w.d*	S370FZDU*w.d*

バイナリデータ読み込み用の SAS プログラムを記述する場合、そのプログラムが 1 種類のコンピュータ上でしか実行されない場合は、ネイティブモードの入力形式と出力形式を使用できます。それに対し、異なるバイト格納システムを使用する複数のコンピュータ上で実行できる SAS プログラムを記述する場合は、IBM 370 入力形式を使用します。IBM 370 入力形式を使用すると、その形式でデータを読み込み、どのような数値データ格納規格の SAS 環境でも実行可能な SAS プログラムを記述できます。[1] また、IBM 370 入力形式を使用すると、IBM メインフレームのネイティブモードで記述されたデータも読み込むことができます。

注: ローカルなエンコーディング環境以外の環境で作成されたテキストファイルを読み込む場合、EBCDIC システムまたは ASCII システム上で ENCODING=オプションを指定する必要があります。ASCII プラットフォーム上で EBCDIC テキストファイルを読み込む場合、FILENAME ステートメントまたは INFILE ステートメントで ENCODING=オプションを指定することを推奨します。ただし、FILENAME ステートメントまたは INFILE ステートメントで DSD オプションと、DLM=または DLMSTR=オプションのいずれかを使用する場合、ENCODING=オプションが必須となります。これは、これらのオプションでは、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)が必要となるためです。エンコーディング固有の入力形式は、文字フィールドと文字以外のフィールドの両方を含む真のバイナリファイルに使用するために予約されています。

すべての SAS 出力形式および入力形式に関する説明や、数値バイナリデータの出力方法に関する詳細については、*SAS Formats and Informats: Reference* を参照してください。

カラムバイナリデータの読み込み

定義

カラムバイナリデータの記憶域
: カラムバイナリデータは、やや古いデータ形式であり、現在ではあまり使用されていません。ほとんどの SAS System のユーザーにとっては必要ありません。カラムバイナリデータ形式では、データを圧縮することで、1 枚の"仮想的な"パンチカードに 80 項目以上のデータを格納できます。この方式の利点は、同じ領域にデータを

[1] たとえば、IBM 370 入力形式を使用することで、バイナリ整数を含むデータをメインフレームから PC にダウンロードし、そのデータを S370FIB 入力形式を使用して読み込むことができます。

より多く格納できることです。カードイメージのデータセットはまだ利用されることがあるため、SAS System には、カラムバイナリデータを読み込むための入力形式が用意されています。カラムバイナリデータの記憶域の詳細については、“カラムバイナリデータの記憶域の説明” (434 ページ) を参照してください。

カラムバイナリデータの読み込み方法

SAS System を使用してカラムバイナリデータを読み込むには、次の方法があります。

- カラムバイナリデータを読み込む SAS 入力形式を選択する方法
- INFILE ステートメントの RECFM=オプションと LRECL=オプションを設定する方法
- ポインタコントロールを使用する方法

次に、カラムバイナリデータを読み込む SAS 入力形式の一覧表を示します。

表 19.7 *カラムバイナリデータを読み込む SAS 入力形式*

入力形式名	説明
$CB*w*.	カラムバイナリファイルから標準文字データを読み込みます。
CB*w*.	カラムバイナリファイルから標準数値データを読み込みます。
PUNCH.*d*	行がパンチされているかどうかを読み込みます。
ROW*w.d*	カラムバイナリフィールドをカードカラムの下方向に読み込みます。

カラムバイナリデータを読み込むには、INFILE ステートメントに 2 つのオプションを設定する必要があります。

- RECFM=オプションを F (固定)に設定します。
- LRECL=オプションを 160 に設定します。これは、カラムバイナリデータの各カードカラムが、フィールドを読み込む前に 2 バイトに拡張されるためです。

たとえば、カラムバイナリデータをファイルから読み込む場合は、データを読み込む INPUT ステートメントの前に、INFILE ステートメントを次の形式で挿入します。

```
infile file-specification or path-name
  recfm=f
lrecl=160;
```

注: カラムバイナリデータの各カラムが 2 バイトに拡張されても、カラムポインタの位置には影響しません。通常どおり絶対カラムポインタコントロールを使用してください。2 バイトに拡張されたレコードの実際の位置は、入力形式によって自動的に計算されます。値が列 23 にある場合、そこにポインタを移動させるには、ポインタコントロール@23 を使用します。

カラムバイナリデータの記憶域の説明

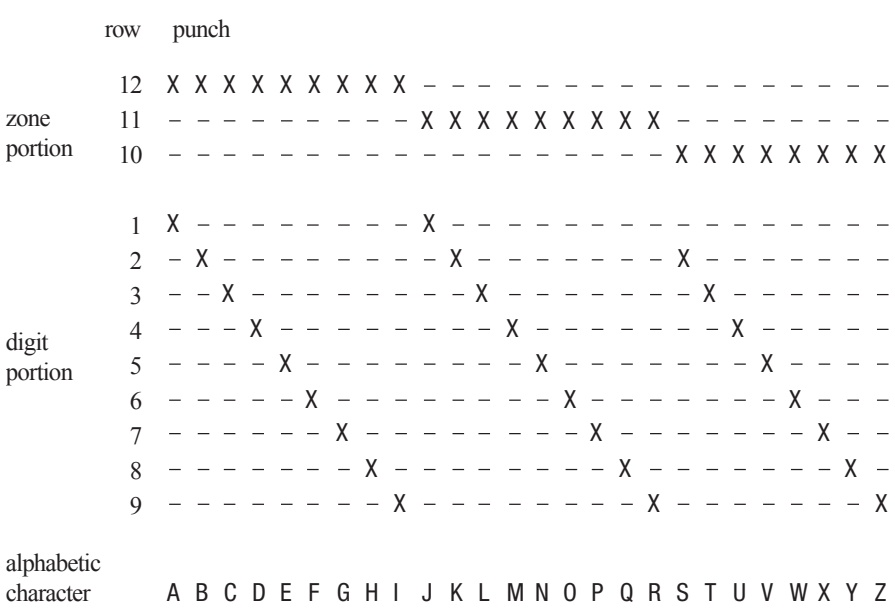
物理的なパンチカード上にあるカラムの行配置と番号付けは、文字と数値をエンコードするための手法であるホレリスシステムに由来します。これは、1 組の値を使用して、1 つの文字または 1 桁の数字を表現するものでした。ホレリスシステムでは、カード上の各カラムには最大で 2 個のパンチ穴が空いていました。これら 2 個のパンチ穴は、1 つはゾーン部に、もう 1 つは数字部に空いており、1 組の値に対応していまし

た。1 組の値のそれぞれには、特定のアルファベット、記号、数字が対応していました。

パンチカードのゾーン部(先頭の 3 行)では、1 組の値を表すためのゾーン部の値は、12、11、0 (または 10)、パンチ穴なしのいずれかになります。パンチカードの数字部は 4 - 12 行目です。1 組の値を表すための数字部の値は、1 - 9、またはパンチ穴なしのいずれかになります。

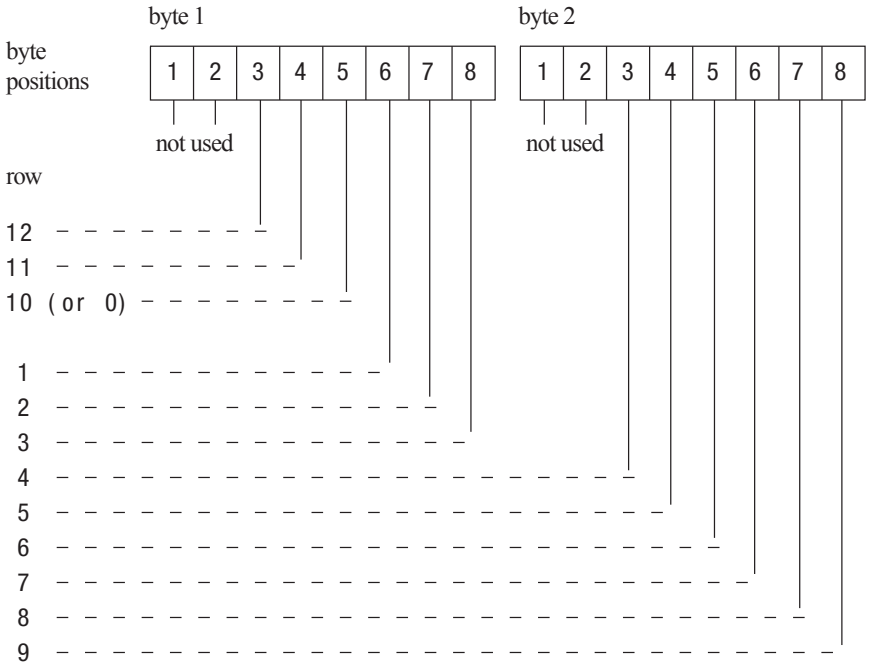
次の図は、アルファベット文字に対応する複数パンチ穴の組み合わせを示しています。

図 19.1 *パンチカードにおけるカラムと行*



SAS System は、カラムバイナリデータの各カラム("仮想的な"パンチカード)に含まれている情報を、2 バイトを使用して格納します。カラムバイナリデータの 1 つのカラムには 12 行までの構成しかありませんが、2 バイトを使用すると 16 行までの位置を保持できます。そのため、2 つのバイトにある使われない 4 ビットの情報は、各バイトの先頭の 2 ビットに配置されます。次の図は、SAS System によるカラムバイナリデータで使用する 2 バイトの位置と"仮想的な"パンチカードデータの行との対応を示しています。SAS System では、パンチ穴が空いている位置をバイナリの 1 ビットとして格納し、空いていない位置をバイナリの 0 ビットとして格納します。

図 19.2 “仮想的な”パンチカードにおけるカラムバイナリデータの表現



20 章

DATA ステップでの BY グループ処理

BY グループ処理の定義

BY グループ処理
1 つ以上の共通する変数の値を基準にしてグループ化または並べ替えた SAS データセットが 1 つ以上ある場合に、そのデータセットにあるオブザベーションをグループごとに処理する方法です。複数の SAS データセットを結合して処理するためには、DATA ステップにおける BY グループ処理が最も一般的に使用されます。これには、SET、MERGE、MODIFY、UPDATE のいずれかのステートメントと BY ステートメントを一緒に使用します。

BY 変数
データセットの並べ替えまたはインデックス作成の基準となる、BY グループ処理に共通する特定の変数です。SET、MERGE、UPDATE のいずれかのステートメントを使用する場合は、すべてのデータセットが、BY 変数の値を基準にして並べ替えまたはインデックス付けされている必要があります。MODIFY ステートメントを使用する場合には、データが並べ替えられている必要はありません。ただし、プログラムでの処理効率は、並べ替えられたデータを処理する方が良くなります。結合するデータセットは、すべて 1 個以上の BY 変数を含んでいる必要があります。オブザベーション内での BY 変数の位置は処理に影響しません。

BY 値
BY 変数の値、または BY 変数のフォーマット指定値です。

BY グループ
共通する BY 値を持つオブザベーションの集まりです。BY ステートメント内で複数の変数を使用する場合、BY グループはそれらの変数の値の組み合わせが同一であるオブザベーションの集まりになります。各 BY グループは、変数の値の組み合わせがそれぞれ一意になります。

FIRST.variable と LAST.variable
は、BY グループの始まりと終わりを識別することができる、BY 変数ごとに作成される変数です。FIRST.*variable* は、BY グループ内の最初のオブザベーションの処理時に 1 に設定されます。LAST.*variable* は、BY グループ内の最後のオブザベーションの処理時に 1 に設定されます。これらの変数を利用すると、BY グループ処理の開始時または終了時にさまざまなアクションを実行できるようになります。詳細については、“DATA ステップでの BY グループ識別” (442 ページ)を参照してください。

BY グループ処理の詳細については、21 章, “SAS データセットの加工” (453 ページ)を参照してください。また、*Combining and Modifying SAS Data Sets:Examples* も参照してください。

BY グループ処理の構文

BY グループ処理は、次のいずれかの構文に従って記述します。

BY *variable(s)*;

BY <DESCENDING> *variable(s)* <NOTSORTED> <GROUPFORMAT>;

各引数の意味は次のとおりです。

DESCENDING
指定されている変数に基づき、降順(大きい値から小さい値の順)で、データセットが並べ替えられることを示します。BY グループ内で複数の変数を使用した場合、DESCENDING オプションの直後に指定した変数だけに、DESCENDING が適用されます。

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。

注: SET、MERGE、UPDATE のいずれかのステートメントを使用して処理する場合は、すべてのデータセットが、BY 変数の値を基準にして並べ替えまたはインデックス付けされている必要があります。MODIFY ステートメントを使用する場合には、データが並べ替えられている必要はありません。ただし、プログラムでの処理効率は、並べ替えられたデータを処理する方が良くなります。結合するデータセットは、共通する 1 個以上の BY 変数を含んでいる必要があります。オブザベーション内での BY 変数の位置は処理に影響しません。

NOTSORTED
同一の BY 値を持つオブザベーションが、グループ化されてはいるものの、必ずしもアルファベット順または数値順には並べ替えられていないことを指定します。

GROUPFORMAT
BY 変数の内部値ではなく、フォーマット指定値を使用して、BY グループの始まりと終わりを決定します(すなわち、変数 FIRST.*variable* および LAST.*variable* にそれぞれ値を割り当てます)。GROUPFORMAT オプションは BY ステートメント内の任意の場所に指定できます。指定した場合、その BY ステートメント内のすべての変数に対してこのオプションが適用されます。

BY ステートメントの完全な説明については、"BY Statement" (*SAS Statements: Reference*)を参照してください。

BY グループについて

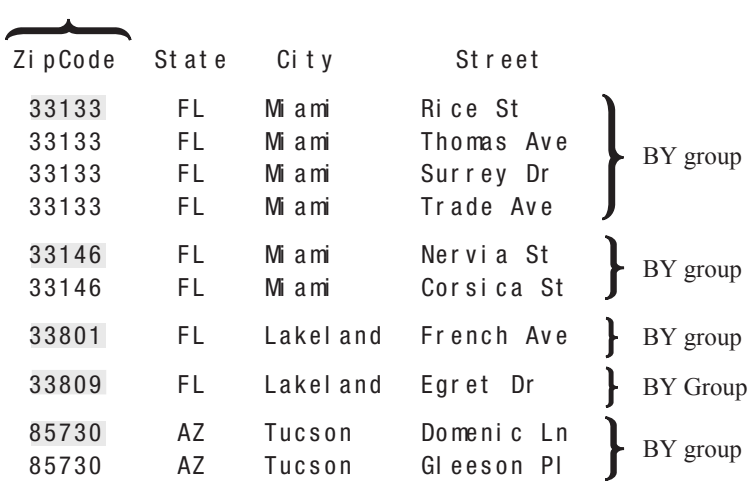
1 つの BY 変数を使用した BY グループ

次の図は、BY 変数 ZipCode を 1 つ指定した場合のデータの処理結果を示しています。入力データセットには、変数 street、city、state、Zipcode が格納されています。このデータセットは、次の BY ステートメントによる順序で並べ替えられています。

```
by ZipCode;
```

下記の図は、BY 変数 ZipCode による 5 つの BY グループを示しています。このデータセットでは、見やすくするために BY 変数 ZipCode の値を左側に示してあります。ただし、オブザベーション内での BY 変数の位置は処理には影響しません。

図 20.1 *1 つの BY 変数(ZipCode)を使用した BY グループ*



最初の BY グループには、BY 値が最も小さい(33133)オブザベーションがすべて含まれています。2 番目の BY グループには、BY 値が 2 番目に小さい(33146)オブザベーションがすべて含まれています。それ以降も同様に、5 つの BY グループが形成されています。

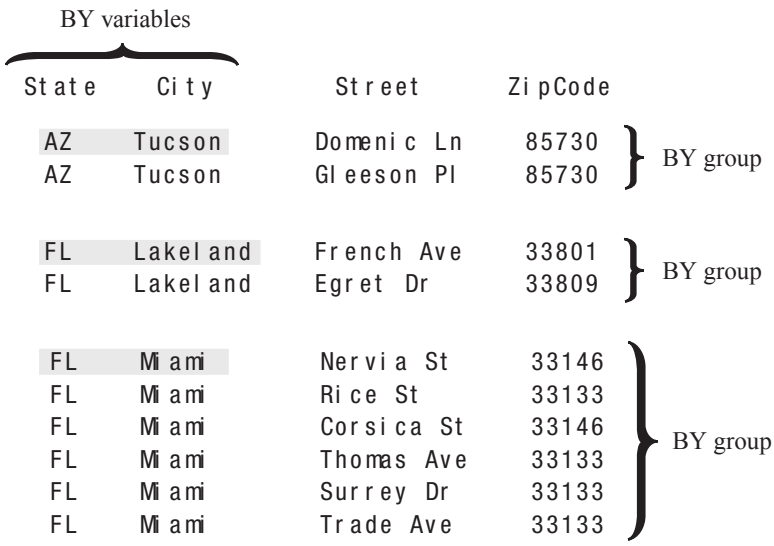
複数の BY 変数を使用した BY グループ

次の図は、State と City という BY 変数を 2 つ指定した場合のデータの処理結果を示しています。この例では、“1 つの BY 変数を使用した BY グループ” (439 ページ)と同じデータセットを使用しています。このデータセットは、次の BY ステートメントによる順序で並べ替えられています。

```
by State City;
```

下記の図には、3 つの BY グループが示されています。このデータセットでは、見やすくするために BY 変数 State と City の値を左側に示してあります。ただし、オブザベーション内での BY 変数の位置は処理には影響しません。

図 20.2 BY 変数 State と City がある BY グループ



このデータセットは、変数 State の値が AZ(アリゾナ)であるオブザベーションが先頭となるように並べ替えられています。BY 変数が State、City の順に指定されているため、変数 State の値が同じである場合、変数 City の値によって BY グループが形成されます。各 BY グループは、変数 State と City の値の組み合わせがそれぞれ一意になっています。この例では、最初の BY グループの BY 値は **AZ Tucson**、2 番目の BY グループの BY 値は **FL Lakeland** になります。

BY グループ処理の呼び出し

DATA ステップと PROC ステップでは、BY ステートメントを使用することで BY グループ処理を呼び出すことができます。たとえば、次の DATA ステップでは、SET ステートメントを使用してファイルを読み込むことにより、3 つの SAS データセットにあるオブザベーションをインタリーブしています。BY ステートメントによって、データの並べ替え方法を指定しています。

```
data all_sales;
   set region1 region2 region3;
   by State City Zip;
   … more SAS statements …
run;
```

ここで説明しているのは、DATA ステップでの BY グループ処理です。プロシジャでの BY グループ処理については、“Creating Titles That Contain BY-Group Information ” (*Base SAS Procedures Guide*)を参照してください。

BY グループ処理の前処理が必要なデータであるかどうかの判定

SET、MERGE、または UPDATE ステートメントでグループ化や並べ替えを行ったデータを使用して、1 つ以上の SAS データセットを処理する場合は、事前にデータを確認して、前処理が必要かどうかを判定する必要があります。データセット内のすべてのオブザベーションが次に示すいずれかの順序に従って並んでいる場合は、データセットを事前に並べ替える前処理は不要です。

- 数値が昇順または降順で並んでいる場合
- 文字値が昇順または降順で並んでいる場合
- アルファベット順や数字順には並んでいないが、暦やフォーマット指定値などの何らかの規則に従ってグループ化された順序で並んでいる場合

オブザベーションが目的の順序で並んでいない場合は、BY グループ処理を使用する前に、データセットを並べ替えるか、インデックスを作成する必要があります。

BY グループ処理で MODIFY ステートメントを使用する場合には、入力データの事前の並べ替えは不要です。ただし、事前に並べ替えることで処理効率が向上します。

BY グループ処理では、PROC SQL ビューを使用できます。詳細については *SAS SQL プロシジャユーザーガイド*を参照してください。

注: SAS/ACCESS ソフトウェアを使用している場合に、SAS プログラムで SAS ビューまたはライブラリ参照名を使用して BY グループ処理を行う方法については、*SAS/ACCESS for Relational Databases: Reference* を参照してください。

BY グループ処理のための入力データの前処理

BY グループ処理のためのオブザベーションの並べ替え

SORT プロシジャを使用すると、データセット内にあるオブザベーションを物理的に並べ替えることができます。SORT プロシジャの OUT=オプションを使用すると、元のデータセットを置き換えることや、並べ替えられた新しいデータセットを作成することができます。この例では、SORT プロシジャを使用して、データセット INFORMATION 内にあるオブザベーションを変数 State と ZipCode の値(昇順)に基づいて並べ替えた後、元のデータセットを置き換えています。

```
proc sort data=information;
   by State ZipCode;
run;
```

SORT プロシジャを使用する場合は、BY ステートメント内で変数を指定する順序を、DATA ステップの BY ステートメントで指定するのと同じ順序にしてください。数値変数と文字変数のデフォルトの並べ替え順序の詳細については、*Base SAS Procedures Guide* にある SORT プロシジャの説明を参照してください。

注: SORT プロシジャで SORTSEQ=LINGUISTIC オプションが指定されている場合、BY ステートメントは並べ替え済みデータの言語照合を尊重します。

BY グループ処理のためのインデックス作成

SAS データセットの 1 つ以上の変数に基づいてインデックスを作成することによっても、オブザベーションを数値または文字の昇順で処理できます。DATA ステップ内で BY ステートメントを指定すると、SAS System は適切なインデックスを探します。インデックスを検出した場合、SAS System は、データセットからオブザベーションを取得する際に自動的にインデックス順序に従います。

注: インデックスの作成と保持には、より大きなシステムリソースが必要です。そのため、インデックスを使用することでパフォーマンスが大幅に向上するかどうかを判断する必要があります。データ値を並べ替える場合、SAS データセットに含まれるデータの性質によっては、インデックスを作成するよりも SORT プロシジャを使用する方が効率がよくなります。インデックスの概要については、"SAS インデックスについて" (632 ページ)を参照してください。

DATA ステップでの BY グループ識別

BY グループ内のオブザベーションの処理

DATA ステップでは、各 BY グループの開始と終了を識別するには、BY 変数ごとに作成される 2 つの一時変数 FIRST.*variable* および LAST.*variable* を使用します。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。これらの一時変数の値により、特定のオブザベーションが次のどれに相当するかを判定できます。

- BY グループの最初のオブザベーション
- BY グループの最後のオブザベーション
- BY グループの最初のオブザベーションでも最後のオブザベーションでもない
- BY グループの最初かつ最後のオブザベーション(BY グループ内にオブザベーションが 1 つしかない)

処理しているオブザベーションが BY グループの最初または最後のものかどうかの条件に応じて、さまざまなアクションを実行できます。

名前リテラルを BY グループ変数として使用する

BY グループ処理の BY 変数として名前リテラルを指定し、対応する FIRST.または LAST.一時変数を参照する場合、2 レベルの変数名の FIRST.または LAST.部分を引用符で囲みます。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;
```

BY グループ処理と、一時変数 FIRST および LAST の作成方法の詳細については、"FIRST.variable と LAST.variable の識別" (443 ページ)および"How SAS Identifies

the Beginning and End of a BY Group" (*SAS Statements: Reference*)を参照してください。

FIRST.variable と LAST.variable の識別

オブザベーションが BY グループ内の最初のオブザベーションである場合、FIRST.*variable* の値が 1 に設定されます。この変数の値は、当該 BY ステートメント内の他の変数と同様に、次のオブザベーションでは変化します。オブザベーションが BY グループ内の最初のオブザベーションではない場合、FIRST.*variable* の値は 0 になります。同様に、オブザベーションが BY グループ内の最後のオブザベーションである場合、LAST.*variable* の値が 1 に設定されます。この変数の値は、当該 BY ステートメント内の他の変数と同様に、次のオブザベーションでは変化します。オブザベーションが BY グループ内の最後のオブザベーションではない場合、LAST.*variable* の値は 0 になります。オブザベーションがデータセット内の最後のオブザベーションである場合、すべての LAST.*variable* 変数の値が 1 に設定されます。

注: SAS 名前リテラルの詳細については、"SAS 名前リテラル" (31 ページ) を参照してください。

例 1:State、City、ZipCode、Street によるオブザベーションのグループ化

次の例では、FIRST.*variable* および LAST.*variable* を使用して、SAS System が 4 つの BY グループ(State、City、ZipCode、Street)の始まりと終わりにフラグを設定する仕組みを示します。プログラムデータベクトル内に、6 つの一時変数が作成されます。これらの変数は、DATA ステップ処理中に使用することはできますが、新しいデータセットの変数としては出力されません。

下記の図に示す SAS データセットでは、オブザベーションは次の BY ステートメントによる順序で並べ替えられています。

```
by State City ZipCode;
```

SAS System は、FIRST.State、LAST.State、FIRST.City、LAST.City、FIRST.ZipCode、LAST.ZipCode という 6 つの一時変数を作成します。

```
options pageno=1 nodate linesize=80 pagesize=60;
data testfile;
   input State $ ZipCode $ City $ Street $ 19-33;
   datalines;
AZ 85730 Tucson   Gleeson Place
FL 33133 Miami    Rice Street
FL 33133 Miami    Thomas Avenue
FL 33133 Miami    Surrey Drive
FL 33146 Miami    Nervia Street
FL 33146 Miami    Corsica Street
OH 45056 Miami    Myrtle Street
;
data test2;
   set testfile;
   by State City ZipCode;
   put _N_= state= first.state= last.state= first.city= last.city=
      first.ZipCode= last.ZipCode= ;
run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

```
79   options pageno=1 nodate linesize=80 pagesize=60;
80   data testfile;
81      input State $ ZipCode $ City $ Street $ 19-33;
82      datalines;
NOTE: The data set WORK.TESTFILE has 7 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

90   ;
91   data test2;
92      set testfile;
93      by State City ZipCode;
94      put _N_= state= first.state= last.state= first.city= last.city=
95         first.ZipCode= last.ZipCode= ;
96   run;
_N_=1 State=AZ FIRST.State=1 LAST.State=1 FIRST.City=1 LAST.City=1
FIRST.ZipCode=1 LAST.ZipCode=1
_N_=2 State=FL FIRST.State=1 LAST.State=0 FIRST.City=1 LAST.City=0
FIRST.ZipCode=1 LAST.ZipCode=0
_N_=3 State=FL FIRST.State=0 LAST.State=0 FIRST.City=0 LAST.City=0
FIRST.ZipCode=0 LAST.ZipCode=0
_N_=4 State=FL FIRST.State=0 LAST.State=0 FIRST.City=0 LAST.City=0
FIRST.ZipCode=0 LAST.ZipCode=1
_N_=5 State=FL FIRST.State=0 LAST.State=0 FIRST.City=0 LAST.City=0
FIRST.ZipCode=1 LAST.ZipCode=0
_N_=6 State=FL FIRST.State=0 LAST.State=1 FIRST.City=0 LAST.City=1
FIRST.ZipCode=0 LAST.ZipCode=1
_N_=7 State=OH FIRST.State=1 LAST.State=1 FIRST.City=1 LAST.City=1
FIRST.ZipCode=1 LAST.ZipCode=1
NOTE: There were 7 observations read from the data set WORK.TESTFILE.
NOTE: The data set WORK.TEST2 has 7 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

97   proc printto; run;
```

表 20.1 *State、City、ZipCode による BY グループ*

4 つの BY グループ内のオブザベーション				**対応する FIRST.と LAST.変数の値**					
State	City	ZipCode	Street	FIRST.State	LAST.State	FIRST.City	LAST.City	FIRST.ZipCode	LAST.ZipCode
AZ	Tucson	85730	Glen Pl	1	1	1	1	1	1
FL	Miami	33133	Rice St	1	0	1	0	1	0
FL	Miami	33133	Tom Ave	0	0	0	0	0	0
FL	Miami	33133	Surrey Dr	0	0	0	0	0	1
FL	Miami	33146	Nervia St	0	0	0	0	1	0

4 つの BY グループ内のオブザベーション				対応する FIRST.と LAST.変数の値					
FL	Miami	33146	Corsica St	0	1	0	1	0	1
OH	Miami	45056	Myrtle St	1	1	1	1	1	1

例 2:City、State、ZIP コード、Street によるオブザベーションのグループ化

次の例では、FIRST.*variable* および LAST.*variable* を使用して、SAS System が 4 つの BY グループ(City、State、ZipCode、Street)の始まりと終わりにフラグを設定する仕組みを示します。プログラムデータベクトル内に、6 つの一時変数が作成されます。これらの変数は、DATA ステップ処理中に使用することはできますが、新しいデータセットの変数としては出力されません。

次の図に示す SAS データセットでは、オブザベーションは次の BY ステートメントによる順序で並べ替えられています。

```
by City State ZipCode;
```

SAS System は、FIRST.City、LAST.City、FIRST.State、LAST.State、FIRST.ZipCode、LAST.ZipCode という 6 つの一時変数を作成します。

表 20.2 City、State、ZIPCode、Street によるオブザベーションのグループ化

4 つの BY グループ内のオブザベーション				対応する FIRST.と LAST.変数の値					
City	State	ZipCode	Street	FIRST.City	LAST.City	FIRST.State	LAST.State	FIRST.ZipCode	LAST.ZipCode
Miami	FL	33133	Rice St	1	0	1	0	1	0
Miami	FL	33133	Tom Ave	0	0	0	0	0	0
Miami	FL	33133	Surrey Dr	0	0	0	0	0	1
Miami	FL	33146	Nervia St	0	0	0	0	1	0
Miami	FL	33146	Corsica St	0	0	0	1	0	1
Miami	OH	45056	Myrtle St	0	1	1	1	1	1
Tucson	AZ	85730	Glen Pl	1	1	1	1	1	1

例 3:FIRST.variable に影響する変更

FIRST.*variable* の値は、その変数の値が変化していない場合であっても、別の変数の値が変化することにより影響を受けることがあります。

次の例では、FIRST.*variable* および LAST.*variable* の値は、BY 変数の値によってだけではなく、並べ替え順序により変化します。オブザベーション 3 を読み込んだ時点で、FIRST.*Y* の値は 1 に設定されます。これは、BLUEBERRY が変数 Y の新しい値であるためです。この Y の変化により、Z の値が変化していない場合であっても、FIRST.*Z* が 1 に設定されます。

```
options pageno=1 nodate linesize=80 pagesize=60;

data testfile;
   input x $ y $ 9-17 z $ 19-26;
   datalines;
apple   banana    coconut
apple   banana    coconut
apple   blueberry citron
apricot blueberry citron
;

data _null_;
   set testfile;
   by x y z;
   if _N_=1 then put 'Grouped by X Y Z';
   put _N_= x= first.x= last.x= first.y= last.y= first.z= last.z= ;
run;

data _null_;
   set testfile;
   by y x z;
   if _N_=1 then put 'Grouped by Y X Z';
   put _N_= x= first.x= last.x= first.y= last.y= first.z= last.z= ;
run;
```

ログ 20.1 *BY 変数の処理結果を示す SAS ログの一部*

```
Grouped by X Y Z _N_=1 x=Apple FIRST.x=1 LAST.x=0 FIRST.y=1 LAST.y=0 FIRST.z=1
LAST.z=0 _N_=2 x=Apple FIRST.x=0 LAST.x=0 FIRST.y=0 LAST.y=1 FIRST.z=0 LAST.z=1
_N_=3 x=Apple FIRST.x=0 LAST.x=1 FIRST.y=1 LAST.y=1 FIRST.z=1 LAST.z=1 _N_=4
x=Apricot FIRST.x=1 LAST.x=1 FIRST.y=1 LAST.y=1 FIRST.z=1 LAST.z=1 Grouped by Y
X Z _N_=1 x=Apple FIRST.x=1 LAST.x=0 FIRST.y=1 LAST.y=0 FIRST.z=1 LAST.z=0 _N_=2
x=Apple FIRST.x=0 LAST.x=1 FIRST.y=0 LAST.y=1 FIRST.z=0 LAST.z=1 _N_=3 x=Apple
FIRST.x=1 LAST.x=1 FIRST.y=1 LAST.y=0 FIRST.z=1 LAST.z=1 _N_=4 x=Apricot
FIRST.x=1 LAST.x=1 FIRST.y=0 LAST.y=1 FIRST.z=1 LAST.z=1
```

DATA ステップでの BY グループ処理

概要

DATA ステップ内での BY グループ処理のうち、最も一般的な使用方法は、SET、MERGE、MODIFY、UPDATE のいずれかのステートメントと BY ステートメントを一緒に使用して、複数の SAS データセットを結合することです。(SET、MERGE、UPDATE ステートメントを BY ステートメントと組み合わせて使用する場合、オブザベーションをグループ化するか、または並べ替える必要があります)。これらのステートメントを処理する場合、SAS System はオブザベーションを一度に 1 つずつプログラムデータベクトル(PDV)に読み込みます。BY グループ処理を使用すると、1 つ以上の BY 変数の値

に基づいてデータセットからオブザベーションを選択し、処理することができます。SAS System は、1 つの BY グループにあるオブザベーションをすべて処理すると、その次の BY グループにあるオブザベーションを処理します。

BY ステートメントは、プログラムデータベクトル内の値を欠損値に設定するタイミングを制御することで、SET、MERGE、MODIFY、UPDATE ステートメントのアクションを変更します。BY グループの処理中、SAS System は、データセット内にある処理対象の BY グループの最後のオブザベーションをコピーするまで、変数の値を保持します。SET ステートメントに BY ステートメントがない場合は、いずれかのデータセットから最後のオブザベーションが読み込まれた時点で、変数が欠損値に設定されます。MERGE ステートメントに BY ステートメントがない場合は、DATA ステップによって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始めた後は、変数は欠損値に設定されません。

BY グループの条件付き処理

オブザベーションを処理するための条件を指定するには、一時変数 FIRST.*variable* および LAST.*variable* (BY グループ処理中に設定される)とともに、サブセット化 IF ステートメントまたは IF-THEN ステートメント、あるいは SELECT ステートメントを使用します。IF ステートメントまたは IF THEN ステートメントを使用すると、たとえば、BY グループの最初または最後の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ときに、各 BY グループのための計算を実行することや、オブザベーションを書き出すことができます。

次の例では、部門別の年間給与支払額を計算しています。IF-THEN ステートメントと、変数 FIRST.*variable* および LAST.*variable* 自動変数を使用して、各 BY グループの先頭で変数 PAYROLL の値を 0 にリセットし、BY グループ内の最後のオブザベーションが処理されたときにオブザベーションを書き出しています。

```
title;

options linesize=80 pagesize=60;

data salaries;
   input Department $ Name $ WageCategory $ WageRate;
   datalines;
BAD Carol Salaried 20000
BAD Elizabeth Salaried 5000
BAD Linda Salaried 7000
BAD Thomas Salaried 9000
BAD Lynne Hourly 230
DDG Jason Hourly 200
DDG Paul Salaried 4000
PPD Kevin Salaried 5500
PPD Amber Hourly 150
PPD Tina Salaried 13000
STD Helen Hourly 200
STD Jim Salaried 8000
;

proc print data=salaries;
run;

proc sort data=salaries out=temp;
   by Department;
run;

data budget (keep=Department Payroll);
```

```
   set temp;
   by Department;
   if WageCategory='Salaried' then YearlyWage=WageRate*12;
   else if WageCategory='Hourly' then YearlyWage=WageRate*2000;
      /* SAS sets FIRST.variable to 1 if this is a new        */
      /* department in the BY group.                          */
   if first.Department then Payroll=0;
   Payroll+YearlyWage;
      /* SAS sets LAST.variable to 1 if this is the last      */
      /* department in the current BY group.                  */
   if last.Department;
run;

proc print data=budget;
   format Payroll dollar10.;
   title 'Annual Payroll by Department';
run;
```

アウトプット 20.1 *条件付き BY グループ処理による出力結果*

Annual Payroll by Department

Obs	Department	Payroll
1	BAD	$952,000
2	DDG	$448,000
3	PPD	$522,000
4	STD	$496,000

アルファベット順や数字順に並んでいないデータ

BY グループ処理では、カレンダ月順やカテゴリ順など、アルファベット順や数字順以外で並んでいるデータを使用できます。その場合、SET ステートメントを使用するときに、BY ステートメントで NOTSORTED オプションを使用します。BY ステートメントに NOTSORTED オプションを指定することで、データがアルファベット順または数値順になってはいないものの、BY 変数の値に基づいてグループ化されていることを通知します。NOTSORTED オプションは、MERGE ステートメントと UPDATE ステートメントでは使用できません。また、SET ステートメントにデータセットが複数指定されているときも使用できません。

次の例では、文字変数 MONTH を基準にしてデータセットがグループ化されているものとします。自動変数 LAST.month の値に基づくサブセット化 IF ステートメントを使用して、条件を満たす場合にオブザベーションを書き出します。この DATA ステップでは、各 BY グループ内の最後のオブザベーションを処理した後にのみ、オブザベーションを書き出しています。

```
data sales;
   input month

data total_sale(drop=sales);
   set region.sales
```

```
   by month notsorted;
   total+sales;
   if last.month;
run;
```

フォーマット値を用いたデータの分類

次のような処理を実行する場合に、BY ステートメントで GROUPFORMAT オプションを使用します。

- DATA ステップ内で FORMAT ステートメントと BY ステートメントの両方が使用されていて、フォーマット指定値を使用してオブザベーションをグループ化する場合
- 変数 FIRST.*variable* および LAST.*variable* を、変数のフォーマット指定値を使用して割り当てる場合

GROUPFORMAT オプションが有効になるのは、SAS データセットを作成する DATA ステップ内で指定された場合だけです。このオプションは、ユーザー定義フォーマットを利用する場合に特に役立ちます。GROUPFORMAT オプションの使用例を次に示します。

例 1:GROUPFORMAT と出力形式の使用

```
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=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 System は次の出力をログに書き出します。

ログ 20.2 *BY ステートメントの GROUPFORMAT オプションを使用した場合の SAS ログ出力*

```
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 2:GROUPFORMAT と出力形式の使用

```
   options
```

```
linesize=80 pagesize=60;

 /* Create SAS data set test  */
data test;
   infile datalines;
   input name $ Score;
datalines;
Jon     1
Anthony 3
Miguel  3
Joseph  4
Ian     5
Jan     6
;
   /* Create a user-defined format */
proc format;
   value Range 1-2='Low'
               3-4='Medium'
               5-6='High';
run;

   /* Create the SAS data set newtest  */
data newtest;
   set test;
   by groupformat Score;
   format Score Range.;
run;

   /* Print using formatted values */
proc print data=newtest;
   title 'Score Categories';
   var Name Score;
   by Score;
run;
```

アウトプット 20.2 BY ステートメントの GROUPFORMAT オプションを使用した場合の SAS 出力

Score Categories

Score=Low

Obs	name	Score
1	Jon	Low

Score=Medium

Obs	name	Score
2	Anthony	Medium
3	Miguel	Medium
4	Joseph	Medium

Score=High

Obs	name	Score
5	Ian	High
6	Jan	High

21 章

SAS データセットの加工

SAS データセットの読み込み、結合、変更

DATA ステップの処理では、SAS データセットの読み込み、結合、変更が行えます。

SAS データセットの読み込み
: SAS データセットを開いて、読み込んだオブザベーションをプログラムデータベクトルへと送ることを意味します。

SAS データセットの結合
: 複数の SAS データセットからデータを読み込み、次のいずれかの結合処理を実行することを意味します。

- 連結
- インタリーブ
- 1 対 1 の読み込み
- 1 対 1 のマージ
- マッチマージ
- データセットの更新

SAS データセットの結合方法の詳細については、"SAS データセットの結合:方法" (466 ページ)を参照してください。

SAS データセットの変更
: MODIFY ステートメントを使用して、SAS データセット内の情報を更新することを意味します。MODIFY ステートメントでは、データセットのコピーを作成せずにデータセットを直接変更するため、ディスク領域を節約できます。SAS データセットの変更は、プログラムステートメント、または別のデータセットに格納されたデータを使用して実行できます。

ツールの概要

SAS データセットの読み込み、結合、変更に使用する主な方法は、SET、MERGE、MODIFY、UPDATE という 4 つのステートメントです。ここでは、これらのステートメントについて、使用例を示しながら説明します。これらのステートメントに関する詳細については、*SAS Statements: Reference* を参照してください。

SAS データセットの読み込み

単一 SAS データセットの読み込み

既存の SAS データセットからデータを読み込むには、SET ステートメントを使用します。次の例の DATA ステップでは、データセット Perm.Tour155_Basic_Cost からデータを読み込み、3 つの新しい変数 Total_Cost、Peak_Cost、Average_Night_Cost の値を計算して、データセット Perm.Tour155_PeakCost を作成しています。

```
data perm.tour155_peakcost;
   set perm.tour155_basic_cost;
   Total_Cost=AirCost+LandCost;
   Peak_Cost=(AirCost*1.15);
   Average_Night_Cost=LandCost/Nights;
run;
```

複数の SAS データセットからの読み込み

複数の SAS データセットから読み込んだデータに対して、さまざまな方法で結合や変更の処理を実行できます。たとえば、複数の入力データセットを結合して 1 つの出力

データセットを作成すること、共通の変数を持つ複数の入力データセットからデータを取り出して結合すること、トランザクションデータセットに基づいてマスタデータセットを更新することなどが行えます。

複数の SAS データセットからの読み込みに関する詳細については、“SAS データセットの結合:方法” (466 ページ)を参照してください。

変数およびオブザベーションの読み込みと書き込みの制御

SAS System では、明示的に命令を与えない限り、変数とオブザベーションはすべて入力データセットから読み込まれ、出力データセットへと書き出されます。SAS ステートメント、データセットオプション、関数を使用して、読み込みまたは書き出しの対象にする変数とオブザベーションを制御できます。次の表に、使用できるステートメントとデータセットオプションの一覧を示します。

表 21.1 読み込みと書き出しを制御するステートメント、データセットオプション、システムオプション

タスク	ステートメント	データセットオプション	システムオプション
変数の制御	DROP	DROP=	
	KEEP	KEEP=	
	RENAME	RENAME=	
オブザベーションの制御	WHERE	WHERE=	FIRSTOBS=
	サブセット化 IF	FIRSTOBS=	OBS=
	DELETE	OBS=	
	REMOVE		
	OUTPUT		

出力データセットに書き出す変数とオブザベーションを制御するには、ステートメントまたはデータセットオプション(KEEP=または DROP=)を使用します。ただし、WHERE ステートメントは例外です。WHERE ステートメントは、変数の値に基づいて、プログラムデータベクトルに書き出すオブザベーションを制御します。入力データセットまたは出力データセットの機能と制御対象に応じて、データセットオプション(WHERE=など)を使用してください。SAS システムオプションを使用してデータを制御することもできます。

SAS データセットの結合:基本概念

複数の SAS データセットに格納されている情報を結合するための前処理の検討

ほとんどのアプリケーションでは、データを処理して有益な情報を得るために、入力データをあらかじめ特定の形式に変換する前処理を実行する必要があります。データは、複数の入力データソースから取得されるため、それぞれ形式が異なっていること

が一般的です。そのため、データ分析やデータに基づくレポート作成を行うには、通常は、データを論理的に関連付けるためのデータ操作が前処理として必要になります。

データ操作(前処理)に必要となる作業は、アプリケーションによって異なります。ただし、データへのアクセス、データの結合、データの処理を行うアプリケーションのすべてに共通している要素もあります。データの出力レイアウトを決定したら、次の作業を行います。

- 入力データをどのように論理的に関連付けるかを決定します。
- データが適切に並べ替えまたはインデックス付けされているかを確認します(必要な場合のみ)。
- 入力データを処理するための適切なアクセス方法を選択します。
- タスクに必要となる適切な SAS System の機能を選択します。

データ間の 4 つの関連付け

データリレーションシップのカテゴリ

入力データソースが複数あり、それらのデータソースが、物理的または論理的なレベルで共通するデータを保持している場合、それらのソースの間にはある関係(リレーションシップ)が存在します。たとえば、従業員データと部門データの場合は、両者が共有する従業員 ID 変数の値を介して関連付けることができます。連続番号を保持するデータセットがある場合には、保持する値の一部分を、オブザベーション番号によって別のデータセットへと論理的に関連付けることができます。

ここで必要な点は、データの間に存在するリレーションシップを識別することです。この点は、入力データを処理して、目的とする出力結果を得る方法を知るために重要となります。データ間の関連付けは、データセット間でオブザベーションを関連付ける方法に応じて、次の 4 つに分類できます。

- 1 対 1
- 1 対多
- 多対 1
- 多対多

目的とする結果を得るためには、オブザベーションがどのように結合されるか、共通する変数の重複する値がどのように扱われるか、共通する変数の値が欠損している場合や不一致の場合にどのように扱われるかを理解する必要があります。また、データセットの並べ替えまたはインデックス作成という前処理をする必要がある場合もあります。各方法の詳細については、“SAS データセットの結合:方法” (466 ページ)を参照してください。

1 対 1 のリレーションシップ

1 対 1 のリレーションシップでは、1 つ以上の共通する変数の値に基づいて、1 つのデータセットにある 1 つのオブザベーションが、別のデータセットにある 1 つのオブザベーションと 1 対 1 に関連付けられます。これは、共通する変数のそれぞれの値が、各データセット内に 1 つしか存在しないことを意味します。複数の変数の場合、1 対 1 のリレーションシップでは、各データセット内で値が重複しないことになります。

次の例では、データセット Salary とデータセット Taxes に含まれるオブザベーションが、共通に持っている変数 EmployeeNumber の値に基づいて関連付けられています。

図 21.1 1 対 1 のリレーションシップ

SALARY

EmployeeNumber	Salary
1234	55000
3333	72000
4876	32000
5489	17000

TAXES

EmployeeNumber	TaxBracket
1111	0.18
1234	0.28
3333	0.32
4222	0.18
4876	0.24

1 対多のリレーションシップと多対 1 のリレーションシップ

入力データセット間で 1 対多または多対 1 のリレーションシップがあるということは、一方のデータセットが、選択した 1 つの変数において特定の値を持つオブザベーションを 0 個または 1 個含み、他方の入力データセットが、それに対応する複数のオブザベーションを含んでいることを意味します。複数の変数を選択して処理する場合、1 対多または多対 1 のリレーションシップでは、1 つのデータセット内で変数の値の組み合わせが重複しないことになります。ただし、その組み合わせが別のデータセット内とは重複する可能性があります。入力データセットの処理順序は、1 対多のリレーションシップであるか、多対 1 のリレーションシップであるかによって決定します。

次の例では、データセット One とデータセット Two に含まれるオブザベーションが、変数 A の共通する値に基づいて関連付けられています。変数 A の値は、データセット One では一意ですが、データセット Two では重複しています。

図 21.2 1 対多のリレーションシップ

ONE

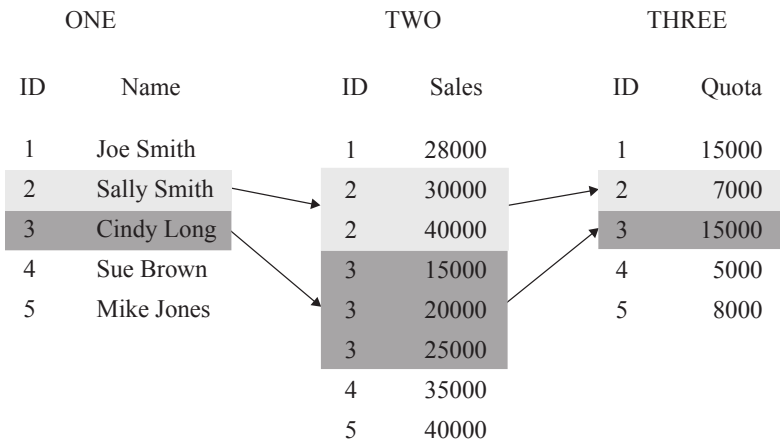
A	B	C
1	5	6
3	3	4

TWO

A	E	F
1	2	0
1	3	99
1	4	88
1	5	77
2	1	66
2	2	55
3	4	44

次の例では、データセット One、Two、Three に含まれるオブザベーションが、共通する変数 ID の値に基づいて関連付けられています。変数 ID の値は、データセット One および Three では一意ですが、データセット Two では重複しています。変数 ID の値 2 と 3 に注目すると、データセット One とデータセット Two のオブザベーション間に 1 対多のリレーションシップが存在し、データセット Two とデータセット Three のオブザベーション間に多対 1 のリレーションシップが存在しています。

図 21.3 1 対多のリレーションシップと多対 1 のリレーションシップ

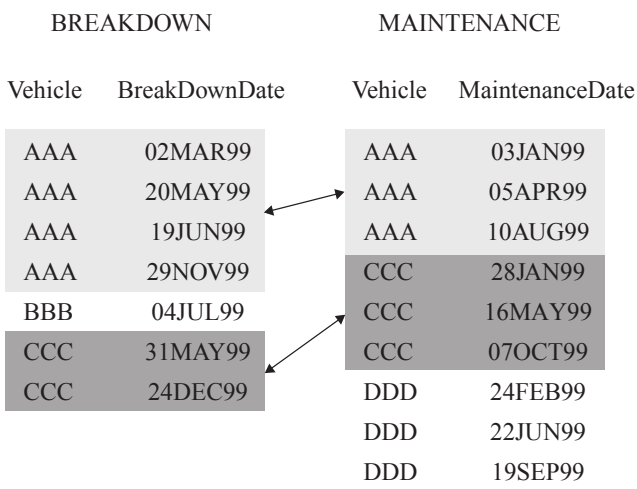


多対多のリレーションシップ

多対多のリレーションシップとは、各入力データセットに含まれている複数のオブザベーションが、1 つ以上の共通する変数の値に基づいて関連付けられていることを意味します。

次の例では、データセット BreakDown とデータセット Maintenance に含まれるオブザベーションが、共通する変数 Vehicle の値に基づいて関連付けられています。Vehicle の値は、各データセット内で一意ではありません。変数 Vehicle の値 AAA と CCC に注目すると、これらのデータセットのオブザベーション間に多対多のリレーションシップが存在しています。

図 21.4 多対多のリレーションシップ



アクセス方式:シーケンシャルアクセスとダイレクトアクセス

概要

データリレーションシップの確立後、次に、そのデータの関連付けに最も適したデータアクセス方法を決定します。オブザベーションには、物理ファイル内でのオブザベーションの配置に従ってシーケンシャルアクセスすることもできれば、ダイレクトアクセスすることもできます。ダイレクトアクセスでは、先行する他のオブザベーションを処理することなく、SAS データセット内の特定のオブザベーションにアクセスします。

シーケンシャルアクセス

DATA ステップを使用してデータを処理する場合、最も簡単で、よく利用される方法は、データセット内のオブザベーションに順番にアクセスして、シーケンシャルに読み込む方法です。オブザベーションをシーケンシャルに読み込むには、SET、MERGE、UPDATE、MODIFY のいずれかのステートメントを使用します。また、OPEN、FETCH、FETCHOBS などの SAS ファイル入出力関数も使用できます。

ダイレクトアクセス

ダイレクトアクセスでは、次の方法で、プログラムから特定のオブザベーションに直接アクセスできます。

- オブザベーション番号に基づいてアクセスする方法
- 単一インデックスまたは複合インデックスを経由して、1 つ以上のキー変数が持つ値に基づいてアクセスする方法

オブザベーション番号に基づいてダイレクトアクセスするには、SET ステートメントまたは MODIFY ステートメントとともに、POINT=オプションを指定します。POINT=オプションには、オブザベーション番号を値として持つ数値変数を指定します。ここで指定した変数の値によって、SET ステートメントまたは MODIFY ステートメントで読み込むオブザベーションが決定します。

1 つ以上の指定変数の値に基づいてオブザベーションにダイレクトアクセスするには、まず変数のインデックスを作成します。その後、SET ステートメントまたは MODIFY ステートメントとともに KEY=ステートメントオプションを指定して、データセットを読み込みます。インデックスとは、1 つ以上のキー変数のデータ値が含まれた、物理的にデータセットとは独立した SAS ファイルです。

注: オブザベーション番号に基づくダイレクトアクセスは、CUROBS、NOTE、POINT、FETCHOBS などの入出力関数を使用して行うこともできます。

SAS データセットの結合方法の概要

SAS データセットの結合方法

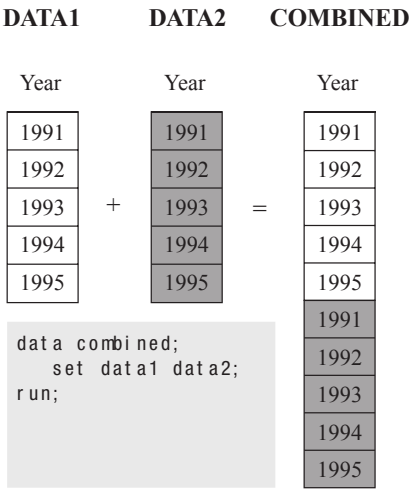
SAS データセットは、次の方法で結合できます。

- 連結
- インタリーブ
- 1 対 1 の読み込み
- 1 対 1 のマージ
- マッチマージ
- 更新

連結

次の図は、2 つの SAS データセットの連結結果を示しています。データセットの連結では、一方のデータセットに含まれているオブザベーションが、もう一方のデータセットに追加されます。この DATA ステップでは、Data1 のすべてのオブザベーションがシーケンシャルに読み込まれ、処理された後、Data2 が読み込まれます。連結した処理の結果は、データセット Combined に格納されます。データセットは、SET ステートメントに列挙された順序で処理されます。

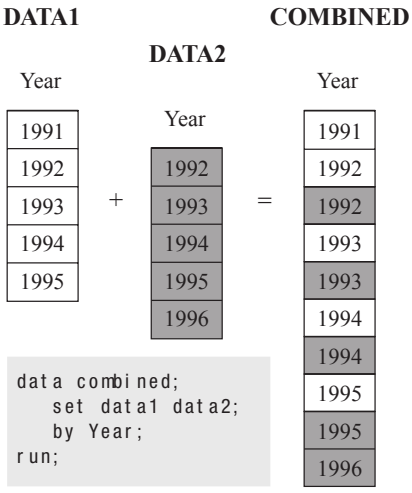
図 21.5 2 つのデータセットの連結



インタリーブ

次の図は、2 つの SAS データセットをインタリーブした結果を示しています。データセットのインタリーブでは、複数のデータセットからオブザベーションを取り出し、共通する 1 つ以上の変数を基準として、別々に格納します。データセット Combined は処理の結果を示しています。

図 21.6 2 つのデータセットのインタリーブ

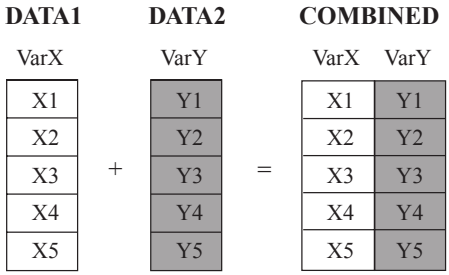


1 対 1 の読み込みと 1 対 1 のマージ

次の図は、1 対 1 の読み込みと 1 対 1 のマージの結果を示しています。1 対 1 の読み込みでは、複数の SAS データセットにある変数をすべて含んだオブザベーションを作成することで、それらの SAS データセットに含まれるオブザベーションを結合します。オブザベーションは、それぞれのデータセット内での相対的な位置に基づいて結合されます。つまり、一方のデータセットの最初のオブザベーションは、もう一方のオブザベーションの最初のオブザベーションと結合されます。以降も同様に処理されます。DATA ステップは、最も小さなデータセットの最後のオブザベーションを読み込んだ後に停止します。1 対 1 のマージは 1 対 1 の読み込みと似ていますが、2 つの点で異なります。1 対 1 のマージでは、複数の SET ステートメントではなく MERGE ステートメントを使用します。さらに、1 対 1 のマージでは、DATA ステップはすべてのデータセット

からすべてのオブザベーションを読み込みます。データセット Combined は処理の結果を示しています。

図 21.7 1 対 1 の読み込みと 1 対 1 のマージ



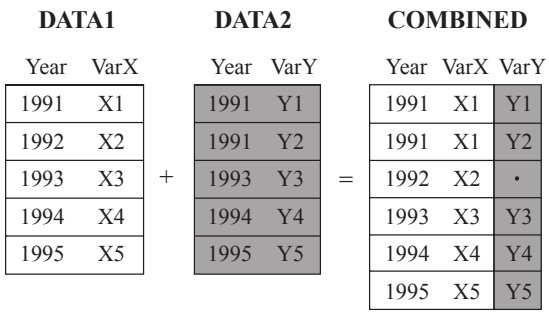

```
data combined;
   set data1;
   set data2;
run;

data combined;
   merge data1 data2;
run;
```

マッチマージ

次の図は、マッチマージの結果を示しています。マッチマージでは、複数の SAS データセットから、共通する 1 つ以上の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。データセット Combined は処理の結果を示しています。

図 21.8 2 つのデータセットのマッチマージ




```
data combined;
   merge data1 data2;
   by Year;
run;
```

更新

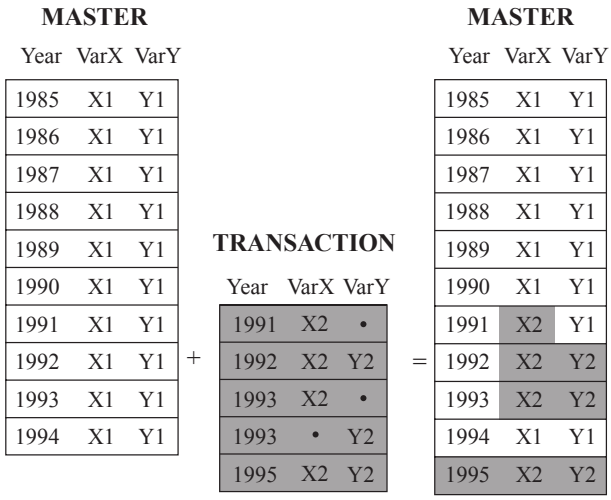
次の図は、マスタデータセットを更新した結果を示しています。更新では、トランザクションデータセット内のオブザベーションから取り出した情報を使用して、マスタデータセット内のオブザベーションに含まれる情報を削除または変更したり、オブザベーションに情報を追加したりします。マスタデータセットを更新するには、UPDATE ステートメントまたは MODIFY ステートメントを使用します。UPDATE ステートメントを使用する場合は、入力データセットが、BY ステートメントに列挙された変数の値に基づいて並べ替えられている必要があります。この例では、データセット Master と Transaction は、

変数 Year を基準として並べ替えられています。MODIFY ステートメントを使用する場合には、データセットが並べ替えられている必要はありません。

UPDATE ステートメントによる更新では、列を追加または削除したり、列の名前を変更したりできます。MODIFY ステートメントによる更新では、変更のあったレコードだけが直接置き換えられます。新しいレコードはファイルの末尾に追加されます。

UPDATE ステートメントと MODIFY ステートメントのデフォルトでは、トランザクションデータセット内にある欠損値によってマスタデータセット内にある非欠損値が置き換えられることはありません。

図 21.9 *マスタデータセットの更新*




```
data master;
   update master transaction;
   by Year;
run;
```

SAS データセットの結合ツールの概要

ステートメントとプロシジャの使用

基本的なデータリレーションシップの確立方法、データへのアクセス方法、SAS データセットを結合する方法について理解すると、データへのアクセス、データの結合、データの処理に使用するさまざまな SAS System の機能を選択できるようになります。次の表に、SAS データセットの結合に使用できるステートメントとプロシジャについて、簡単な説明を記載します。

表 21.2 SAS データセットの結合に使用するステートメントとプロシジャ

		アクセス方法			
ステートメントまたはプロシジャ	実行されるアクション	シーケンシャルアクセス	ダイレクトアクセス	BY ステートメントとの併用	コメント
BY ステートメント	SET、MERGE、MODIFY、UPDATE ステートメントについて、グループ処理を実行します。	NA	NA	NA	グループ処理は、1 つ以上の共通する変数に基づいて、BY 変数の値が同じオブザベーションをグループごとに処理する方法です。
MERGE ステートメント	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、1 つのオブザベーションにします。	X		X	BY ステートメントと併用する場合、BY 変数に基づいてデータセットが並べ替えまたはインデックス付けされている必要があります。
MODIFY ステートメント	SAS データセット内のオブザベーションをダイレクトアクセスして処理します。(UPDATE ステートメントとは異なります)。	X	X	X	BY ステートメントと併用する場合、データセットの並べ替えまたはインデックスを作成することでパフォーマンスが向上します。
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。	X	X	X	データにダイレクトアクセスするには、KEY=または POINT=オプションを使用します。
UPDATE ステートメント	マスタデータセット内のオブザベーションにトランザクションデータセットを適用して更新します。オブザベーションを直接処理せず、データセットのコピーを作成して更新します。	X		X	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、BY 変数に基づいて、並べ替えまたはインデックス付けされている必要があります。
PROC APPEND	1 つの SAS データセットからオブザベーションを読み込み、別の SAS データセットの末尾に追加します。	X			
PROC SQL*	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。最大 32 個までの SAS データセット(テーブル)を読み込んで結合し、1 つのオブザベーションにします。	X	X	X	SQL プロシジャでは 3 つのアクセス方法をすべて利用できますが、アクセス方法は内部の最適化処理によって選択されます。

* SQL プロシジャは、SAS System における構造化照会言語(SQL)を実装したプロシジャです。SQL プロシジャは、SQL が持つ機能に加えて、出力形式や SAS マクロ言語の使用など、SAS System 固有の機能を備えています。

エラーチェックの使用

DATA ステップ内で_IORC_自動変数と SYSRC 自動呼び出しマクロを使用すると、エラーチェックを実行できます。これらの機能は、MODIFY ステートメントとともに、または SET ステートメントで KEY=オプションとともに使用します。これらのステートメントに関する詳細については、“インデックスを用いたデータのランダムなアクセスまたは更新時のエラーチェック” (497 ページ)を参照してください。

データセットの準備方法

データセットの準備ガイドライン

SAS データセットを結合して目的とする処理結果を得るためには、次のガイドラインに示すようなデータセットの準備が必要です。

- データセットの構造と内容を把握します。
- 発生しがちな問題がないかを調べておきます。
- オブザベーションが正しい順序で配置されていること、またはインデックスを使用して適切な順序で取り出せることを確認します。
- 作成したプログラムをテストします。

データセットの構造と内容を理解する

データの関連付け方法を決定するためには、データセットの構造を把握する必要があります。データセットの構造を参照するには、DATASETS プロシジャまたは CONTENTS プロシジャを実行するか、SAS **エクスプローラ**ウィンドウからプロパティを表示します。プロパティでは、各データセット内にあるオブザベーションの数、各変数の変数名と属性、拡張属性のアルファベット順リスト(データセットと変数の拡張属性を含む)、インデックスとインデックス属性のリストなどのディスクリプタ情報を参照できます。オブザベーションを出力して確認するには、PRINT プロシジャまたは REPORT プロシジャを使用します。

特定のディスクリプタ情報は、VTYPE、VLENGTH、VLENGTHX などの SAS 関数を使用しても表示できます。これらの関数に関する詳細については、*SAS Functions and CALL Routines: Reference* を参照してください。

よくある問題の原因を調べる

プログラムが正しく動作しない場合は、入力データに次のエラーがないかどうかを確認してください。

- 異なるデータセットで、同一の変数名を使用している場合

  SAS System では、新しいデータセットの中に、指定した変数名で保持できる変数は 1 つだけです。2 つのデータセットをマージする場合、同一の変数名で異なるデータを持つ変数が含まれていると、最後のデータセットから読み込まれた値によって一方のデータセットの値が上書きされます。

  このエラーを修正するには、データを結合する前に変数名を変更します。変更するには、SET ステートメント、UPDATE ステートメント、または MERGE ステートメントで RENAME=データセットオプションを使用します。または、DATASETS プロシジャを使用します。

- 異なるデータセットで、共通する変数の属性が異なる場合

  この場合の取り扱い方法は、どの属性が異なるのかによって決まります。

  - 種類属性

変数の種類が異なる場合、DATA ステップの処理が停止し、変数に互換性がないというエラーメッセージが表示されます。

このエラーを修正するには、DATA ステップを使用して変数を作成し直す必要があります。使用する SAS ステートメントは、変数の種類に応じて異なります。

- 長さ属性

  変数の長さが異なる場合、変数の長さは、該当する変数を含んだ最初のデータセットから取得されます。次の例では、MERGE ステートメントに列挙されたデータセットが、すべて変数 Mileage を保持しています。データセット Quarter1 では変数 Mileage の長さは 4 バイトです。データセット Quarter2 では 8 バイト、Quarter3 と Quarter4 では 6 バイトになっています。出力データセット Yearly では、変数 Mileage の長さは 4 バイトになります。これは Quarter1 から取得された長さが 4 バイトであるためです。

  ```
  data yearly;
     merge quarter1 quarter2 quarter3 quarter4;
     by Account;
  run;
  ```

  長さ属性を独自に設定するには、SET、MERGE、UPDATE ステートメントの前に LENGTH ステートメントを記述し、適切な長さを指定します。

  注: データセットを結合した結果として変数の長さが変化した場合、SAS ログに警告メッセージが出力され、ゼロ以外のリターンコードが返されます (例:z/OS の場合、SYSRC=4)。文字値の末尾から意味のないブランクを取り除きたい場合などに、データの切り捨てを行うと、SAS ログに警告が出力され、ゼロ以外のリターンコードが返されます。このような場合に警告をオフにするには、VARLENCHK システムオプションで値 NOWARN を設定します。詳細については、"VARLENCHK= System Option" (*SAS System Options: Reference*)を参照してください。

- ラベル、出力形式、入力形式

  データの属性が異なる場合、属性は該当する変数を含んだ最初のデータセットから取得されます。ただし、ラベル、出力形式、入力形式を明示的に指定した場合はデフォルトが上書きされます。すべてのデータセット内で属性が明示的に指定されている場合は、最初のデータセット内で指定された属性が他のデータセットの属性を上書きします。新しい出力データセットの属性をあらかじめ設定しておくには、ATTRIB ステートメントを使用します。

  変数の属性情報へのアクセスには、VLABEL、VLABELX などの変数情報関数も使用できます。これらの関数に関する詳細については、*SAS Functions and CALL Routines: Reference* を参照してください。

- 拡張属性

  出力形式やラベルと同様に、拡張属性は入力データセットから出力データセットに DATA ステップで自動的に渡されます。拡張属性を含む 2 つの入力データセットを結合する場合、変数を読み込む、プログラムで最初に記述されているデータセットの拡張属性が保持され、その属性が出力データセットに適用されます。新しい出力データセットの拡張属性をあらかじめ設定しておくには、DATASETS プロシジャを使用して、拡張属性の追加、削除、設定、更新を行います。DATASETS プロシジャの詳細については、"Extended Attributes" (*Base SAS Procedures Guide*)を参照してください。

正しい順序の確認

UPDATE ステートメント、SET ステートメント、MERGE ステートメント内で BY グループ処理を使用してデータセットを結合する場合は、データセット内のオブザベーションを、BY ステートメントで列挙した変数の順序であらかじめ並べ替えておきます。または、

データセットに適切なインデックスを定義しておくことが必要です。MODIFY ステートメント内で BY グループ処理を使用する場合には、データセットを並べ替える必要はありません。ただし、データセットを並べ替えることにより処理効率は向上します。BY 変数は両方のデータセットに共通する変数を指定しなければなりません。さらに、変数の属性も同じ属性を指定する必要があります。詳細については、20 章, “DATA ステップでの BY グループ処理” (437 ページ)を参照してください。

プログラムのテスト

データセットを結合するための準備としての最後の段階は、プログラムをテストすることです。サンプルデータとして一部のオブザベーションを含んだ小さな SAS データセットを作成して、プログラムのロジックをすべてテストします。ロジックに誤りがあり、予期しない結果が出力された場合は、DATA ステップデバッガを使用してプログラムをデバッグすることもできます。DATA ステップデバッガの詳細については、*SAS データセットオプション: リファレンス*を参照してください。

SAS データセットの結合:方法

連結

定義

データセットの連結とは、複数のデータセットを結合して、1 つのデータセットを作成する処理です。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、元のデータセットに含まれるオブザベーションの数の合計になります。オブザベーションはシーケンシャルに配置されます。最初のデータセットにあるすべてのオブザベーションが読み込まれるのに続いて、2 番目のデータセットにあるオブザベーションがすべて読み込まれます。以降も同様に読み込まれます。

最も単純な結合は、すべてのデータセットが同じ変数を保持している場合です。複数の入力データセットが異なる変数を保持していて、一方のデータセット内でのみ定義されている変数がある結合の場合、もう一方のデータセットに含まれるオブザベーションでは、その変数の値は欠損値になります。どちらの場合も、新しいデータセットに含まれる変数は古いデータセットに含まれる変数と同じになります。

構文

データセットを連結するには、SET ステートメントを次の形式で使用します。

SET *data-set(s)*;

各引数の意味は次のとおりです。

data-set
有効な SAS データセットの名前を指定します。

有効な SAS データセット名に関する詳細については、*SAS Statements: Reference* にある SET ステートメントの説明を参照してください。

データセット連結時の DATA ステップ処理

コンパイルフェーズ
SET ステートメント内に指定された各データセットのディスクリプタ情報が読み込まれ、プログラムデータベクトル(PDV)が作成されます。この PDV には、DATA ステップによって作成された変数とともに、すべてのデータセットから読み込まれた変数が格納されます。

実行: ステップ 1
最初のデータセットに含まれる最初の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ます。最初のオブザベーションが処理され、DATA ステップ内の他のステートメントが実行されます。次に、プログラムデータベクトルの内容が新しいデータセットに書き出されます。

SET ステートメントでは、DATA ステップの処理中に計算されたり割り当てられたりした場合を除き、プログラムデータベクトル内の値が欠損値にリセットされることはありません。DATA ステップにより作成される変数は、そのデータステップの反復が開始されるたびに欠損値へとリセットされます。これに対して、データセットから読み込まれる変数は欠損値へとリセットされません。

実行: ステップ 2
ファイル終端インジケータが検出されるまで、最初のデータセットからオブザベーションが一度に 1 つずつ読み込まれます。プログラムデータベクトル内の変数の値が欠損値に設定された後、2 番目のデータセットからのオブザベーション読み込みが開始されます。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るまで、読み込みが続けられます。

例 1: データセットの連結: DATA ステップを使用した場合

この例では、各データセットに変数 Common と Number が含まれ、オブザベーションは変数 Common の値順に配置されています。データセットの連結は、同じ変数を持つデータセットどうしで行うのが普通です。この例では、データセット結合の効果を分かりやすく示すために、各データセットには一意の変数も格納されています。ライブラリ参照名 Example を使用して参照されるライブラリに含まれている、入力データセット Animal と Plant を次に示します。

Animal

OBS	Common	Animal	Number
1	a	Ant	5
2	b	Bird	
3	c	Cat	17
4	d	Dog	9
5	e	Eagle	
6	f	Frog	76

Plant

OBS	Common	Plant	Number
1	g	Grape	69
2	h	Hazelnut	55
3	i	Indigo	.
4	j	Jicama	14
5	k	Kale	5
6	l	Lentil	77

次のプログラムは、SET ステートメントを使用してデータセットを連結し、その結果を出力します。

```
data concatenation;
   set animal plant;
run;

proc print data=concatenation;
   var Common Animal Plant Number;
   title 'Data Set CONCATENATION';
run;
```

アウトプット 21.1 *連結されたデータセット(DATA ステップを使用)*

Data Set CONCATENATION

Obs	Common	Animal	Plant	Number
1	a	Ant		5
2	b	Bird		.
3	c	Cat		17
4	d	Dog		9
5	e	Eagle		.
6	f	Frog		76
7	g		Grape	69
8	h		Hazelnut	55
9	i		Indigo	.
10	j		Jicama	14
11	k		Kale	5
12	l		Lentil	77

出力データセット CONCATENATION には 12 個のオブザベーションが含まれています。これは、結合に使用したデータセットにあるオブザベーションの合計です。プログラムデータベクトルには、すべてのデータセットの変数が保持されています。一方のデータセットに存在し、もう一方のデータセットには存在していない変数の値は、欠損値に設定されます。

例 2: データセットの連結: SQL を使用した場合

テーブルの連結は、SQL 言語を使用して行うこともできます。この例では、SQL プロシジャによって 2 つのテーブルに含まれる各行を読み込み、Combined という新しいテーブルを作成します。入力テーブル YEAR1 と YEAR2 を次に示します。

```
YEAR1                YEAR2

Date1                Date2

2009
2010                 2010
2011                 2011
2012                 2012
                     2013
                     2014
```

次のプログラムは、入力した 2 つのテーブルを結合してテーブル Combined を作成し出力します。

```
proc sql;
   title 'SQL Table Combined';
```

```
      create table combined as
         select * from year1
         union all
         select * from year2;
         select * from combined;
   quit;
```

アウトプット 21.2 *連結されたテーブル(SQL を使用)*

SQL Table COMBINED

Year
2009
2010
2011
2012
2010
2011
2012
2013
2014

ファイルの追加

データセットまたはテーブルを連結するかわりに、それらを追加することで連結と同じ結果を得ることもできます。SAS System では、データの各レコードを読み込んで新しいファイルを作成することによって、データセット(DATA ステップを使用)またはテーブル(SQL を使用)を連結します。データセットのレコードをすべて読み込まなくてもすむようにするには、APPEND プロシジャを使用して、base=オプションで指定したデータセットに data=オプションで指定したデータセットを追加します。

```
proc append base=year1 data=year2;
run;
```

作成したデータセット YEAR1 には、2 つのデータセットから読み込まれたレコードがすべて含まれます。

注: APPEND プロシジャでは、シーケンシャルライブラリ内の SAS データセットにオブザベーションを追加することはできません。

効率

データセットを連結する場合、他の処理をする必要が特にないときは、DATA ステップを使用するよりも、PROC APPEND を使用するか、PROC DATASETS で APPEND ステートメントを使用する方が処理は効率的です。

インタリーブ

定義

データセットのインタリーブとは、SET ステートメントと BY ステートメントを使用して複数のデータセットを結合し、新しいデータセットを作成する処理です。新しいデータセット内のオブザベーションの数は、元のデータセットに含まれるオブザベーション数の合計になります。新しいデータセット内のオブザベーションは、BY 変数の値に基づいて配置されます。BY グループ内のオブザベーションは、指定したデータセットの順序に基づいて配置されます。データセットのインタリーブを行うには、BY 変数を使うか、またはインデックスを使用します。

構文

BY 変数を使用してデータセットのインタリーブを行うには、SET ステートメントを次の形式で使用します。

SET *data-set(s)*;

BY *variable(s)*;

各引数の意味は次のとおりです。

data-set
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。

variable
データセットの並べ替え基準となる特定の変数を指定します。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。

インデックスを使用してデータセットのインタリーブを行うには、SET ステートメントを次の形式で使用します。

SET *data-set-1 ... data-set-n* KEY= *index*;

各引数の意味は次のとおりです。

data-set
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。

index
インデックス名を指定します。インデックス変数またはキー変数の値に基づいて、シーケンシャルアクセスを使用せずに、SAS データセット内のオブザベーションを読み込みます。

SET ステートメントの KEY=オプションの詳細については、*SAS Statements: Reference* にある SET ステートメントの説明を参照してください。

並べ替えが必要

データセットのインタリーブを行うには、BY ステートメントで使用する変数と同一の変数に基づいてあらかじめオブザベーションを並べ替えておくかグループ化しておく必要があります。または、データセットに適切なインデックス定義が必要です。

インタリーブ時の DATA ステップ処理

コンパイルフェーズ

- SET ステートメント内に指定された各データセットのディスクリプタ情報が読み込まれ、プログラムデータベクトル(PDV)が作成されます。この PDV には、DATA ステップによって作成された変数とともに、すべてのデータセットから読み込まれた変数が格納されます。

- BY ステートメントに指定した変数ごとに、自動変数 FIRST.*variable* および LAST.*variable* が作成されます。

実行: ステップ 1

SET ステートメントに指定された各データセットの最初のオブザベーションが比較され、新しいデータセットの先頭に配置される BY グループが決定されます。選択されたデータセットの最初の BY グループから、オブザベーションがすべて読み込まれます。この BY グループに複数のデータセットが含まれている場合、データセットの読み込み順序は、SET ステートメント内にデータセットを指定したときの順序になります。プログラムデータベクトル内の変数の値は、SAS System が新しいデータセットを読み込むときと、BY グループが変更されるときに欠損値に設定されます。

実行: ステップ 2

各データセットから次のオブザベーションが読み込まれて比較され、次の BY グループが決定されます。この BY グループのオブザベーションが格納されている、SET ステートメント内の選択されたデータセットから、オブザベーションの読み込みが開始されます。すべてのデータセットからオブザベーションが読み込まれるまで、読み込みが続けられます。

例 1: データセットのインタリーブ: 最も単純な場合

この例では、各データセットに共通する BY 変数 Common が含まれ、オブザベーションはその BY 変数の値順に配置されています。ライブラリ参照名 Example を使用して参照されるライブラリに含まれている、入力データセット Animal と Plant を次に示します。

Animal

OBS	Common	Animal
1	a	Ant
2	b	Bird
3	c	Cat
4	d	Dog
5	e	Eagle
6	f	Frog

Plant

OBS	Common	Plant
1	a	Apple
2	b	Banana
3	c	Coconut
4	d	Dewberry
5	e	Eggplant
6	f	Fig

次のプログラムは、SET ステートメントと BY ステートメントを使用してデータセットをインタリーブした結果を出力します。

```
data interleaving;
   set animal plant;
   by Common;
run;

proc print data=interleaving;
   title 'Data Set INTERLEAVING';
run;
```

アウトプット 21.3 インタリーブ後のデータセット

Data Set INTERLEAVING

Obs	Common	Animal	Plant
1	a	Ant	
2	a		Apple
3	b	Bird	
4	b		Banana
5	c	Cat	
6	c		Coconut
7	d	Dog	
8	d		Dewberry
9	e	Eagle	
10	e		Eggplant
11	f	Frog	
12	f		Fig

出力データセット INTERLEAVING には 12 個のオブザベーションが含まれています。これは、結合に使用されたデータセットにあるオブザベーションの合計です。新しいデータセットには、両方のデータセットのすべての変数が格納されています。一方のデータセットにあってもう一方にはない変数の値は、欠損値に設定されます。オブザベーションは BY 変数の値に基づいて配置されます。

例 2: データセットのインタリーブ: BY 変数の値が重複している場合

BY 変数の値がデータセット内で重複している場合、オブザベーションは、読み込み元のデータセットでの格納順序に従って新しいデータセットに書き出されます。次の例では、BY 変数 Common の値が重複しています。入力データセット Animal1 と Plant1 の内容は次のとおりです。

```
     Animal1                        Plant1

OBS  Common  Animal1          OBS  Common  Plant1

 1     a      Ant              1     a     Apple
 2     a      Ape              2     b     Banana
 3     b      Bird             3     c     Coconut
 4     c      Cat              4     c     Celery
 5     d      Dog              5     d     Dewberry
 6     e      Eagle            6     e     Eggplant
```

次のプログラムは、SET ステートメントと BY ステートメントを使用してデータセットをインタリーブした結果を出力します。

```
data interleaving2;
   set animal1 plant1;
   by Common;
```

```
run;

proc print data=interleaving2;
   title 'Data Set INTERLEAVING2: Duplicate BY Values';
run;
```

アウトプット 21.4 *インタリーブ後のデータセット(BY 変数の値が重複している場合)*

Data Set INTERLEAVING2: Duplicate BY Values

Obs	Common	Animal1	Plant1
1	a	Ant	
2	a	Ape	
3	a		Apple
4	b	Bird	
5	b		Banana
6	c	Cat	
7	c		Coconut
8	c		Celery
9	d	Dog	
10	d		Dewberry
11	e	Eagle	
12	e		Eggplant

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、読み込み元のデータセットに含まれるオブザベーションの数の合計になります。オブザベーションが新しいデータセットに書き出される順序は、元のデータセットでのオブザベーションの配置順と同じです。

例 3:データセットのインタリーブ: BY 変数の値が異なる場合

データセット Animal2 とデータセット Plant2 は、一方のデータセットにあって他方のデータセットにはない値を保持しています。入力データセット Animal2 と Plant2 の内容は次のとおりです。

```
    Animal2                         Plant2

OBS  Common  Animal2          OBS  Common  Plant2

 1     a      Ant              1     a     Apple
 2     c      Cat              2     b     Banana
 3     d      Dog              3     c     Coconut
 4     e      Eagle            4     e     Eggplant
                               5     f     Fig
```

次のプログラムは、SET ステートメントと BY ステートメントを使用してこれらのデータセットをインタリーブした結果を出力します。

```
data interleaving3;
   set animal2 plant2;
   by Common;
run;

proc print data=interleaving3;
   title 'Data Set INTERLEAVING3: Different BY Values';
run;
```

アウトプット 21.5 *インタリーブ後のデータセット(BY 変数の値が異なる場合)*

Data Set INTERLEAVING3: Different BY Values

Obs	Common	Animal2	Plant2
1	a	Ant	
2	a		Apple
3	b		Banana
4	c	Cat	
5	c		Coconut
6	d	Dog	
7	e	Eagle	
8	e		Eggplant
9	f		Fig

出力データセットには、BY 変数の値に基づいて配置された 9 個のオブザベーションが格納されています。

コメントと比較

- SAS System 以外の一部のプログラミング言語では、マージという用語がインタリーブの意味で使われることがよくあります。SAS System では、マージという用語は、複数のデータセットから読み込んだオブザベーションを結合して、1 つのオブザベーションにするという意味でのみ使用しています。インタリーブ後のデータセット内にあるオブザベーションは、直接結合されたものではありません。これらのオブザベーションは、元のデータセットから、BY 変数の値の順序に従ってコピーされたものです。
- 1 つのデータセットの中に同じ BY 値を持つ行が複数ある場合、DATA ステップの実行結果では、これらの行の順序が維持されます。
- DATA ステップを使用する場合、入力データセットが適切に並べ替えまたはインデックス付けされている必要があります。SQL プロシジャを使用する場合、入力テーブルが並べ替えられている必要はありません。

1 対 1 の読み込み

定義

1 対 1 の読み込みとは、複数の SET ステートメントを使用して各データセットからオブザベーションを読み込むことによって、複数のデータセットにあるオブザベーションを結合し、1 つのオブザベーションにする処理です。この処理は、1 対 1 のマッチともいいます。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。データセットに他のデータセットと共通する変数が含まれている場合は、最後に読み込まれたデータセットの値によって、それ以前に読み込まれたデータセットの値が置き換えられます。

構文

1 対 1 の読み込みを実行するには、SET ステートメントを次の形式で使用します。

SET *data-set-1*;

SET *data-set-2*;

各引数の意味は次のとおりです。

data-set
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。dataset1 は、DATA ステップが最初に読み込むデータセットです。

data-set-2
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。dataset2 は、DATA ステップが 2 番目に読み込むデータセットです。

注意:
SET ステートメントを複数使用してデータセットを結合する際は、十分に注意が必要です。 複数の SET ステートメントを使用してオブザベーションを結合すると、望ましくない出力結果になる可能性があります。この方法でデータセットを結合する場合は、サンプルデータを用意して、事前にプログラムをテストすることをお勧めします。

詳細については、*SAS Statements: Reference* にある SET ステートメントの説明を参照してください。

1 対 1 の読み込み時の DATA ステップ処理

コンパイルフェーズ
SET ステートメント内に指定された各データセットのディスクリプタ情報が読み込まれ、プログラムデータベクトル(PDV)が作成されます。この PDV には、DATA ステップによって作成された変数とともに、すべてのデータセットから読み込まれた変数が格納されます。

実行: ステップ 1
最初の SET ステートメントが実行されると、最初のデータセットに含まれる最初の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ます。2 番目の SET ステートメントが実行されると、2 番目のデータセットに含まれる最初の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ます。両方のデータセットに同じ変数が含まれている場合は、2 番目のデータセットから読み込まれた値によって、最初のデータセットからの値が置き換えられます。2 番目のデータセットからの値が欠損値であっても同様です。最後のデータセットの最初のオブザベーションが読み込まれ、DATA ステップ内の他のステートメントがすべて実行されると、プログラムデータベクトルの内容が新しいデータセットに書き出されます。SET ステートメン

トでは、DATA ステップの処理中に作成されたり割り当てられたりした場合を除き、プログラムデータベクトル内の値に欠損値が設定されることはありません。

実行: ステップ 2
1 つのデータセットからの読み込みが終わると、他のデータセットからの読み込みが実行されます。読み込みは、いずれかのデータセットでファイル終端インジケータが検出されるまで続けられます。SAS System による処理は、最も小さな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停止します。より大きなデータセットの残りのオブザベーションは、読み込まれません。

例 1:1 対 1 の読み込み:オブザベーションの数が等しい場合

SAS データセット Animal と Plant は、どちらも変数 Common を保持しており、Common の値に基づいて配置されています。入力データセット Animal と Plant の内容は次のとおりです。

```
     Animal                    Plant

OBS  Common  Animal       OBS  Common  Plant

 1     a     Ant           1     a     Apple
 2     b     Bird          2     b     Banana
 3     c     Cat           3     c     Coconut
 4     d     Dog           4     d     Dewberry
 5     e     Eagle         5     e     Eggplant
 6     f     Frog          6     g     Fig
```

次のプログラムは、2 つの SET ステートメントを使用して Animal と Plant からオブザベーションを読み込んで、結合した結果を出力します。

```
data twosets;
   set animal;
   set plant;
run;

proc print data=twosets;
   title 'Data Set TWOSETS - Equal Number of Observations';
run;
```

アウトプット 21.6 *2 つのデータセットから作成されたデータセット(オブザベーションが等しい場合)*

Data Set TWOSETS - Equal Number of Observations

Obs	Common	Animal	Plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	Dewberry
5	e	Eagle	Eggplant
6	g	Frog	Fig

新しいデータセットの各オブザベーションには、入力データセットのすべての変数が格納されます。ただし、ここでオブザベーション 6 の変数 Common の値が“g”であることに注意してください。Animal データセットのオブザベーション 6 の Common の値は、最後に読み込まれたデータセットである Plant の値によって上書きされました。

コメントと比較

- 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、1 対 1 の読み込みでは、データセット内のオブザベーションの数が等しくない場合、すべてのデータセットのオブザベーションを読み込む前に処理が停止します。
- 複数の SET ステートメントを他の DATA ステップステートメントと併用すると、次のことも実行できます。
  - 1 つのオブザベーションに対して複数のオブザベーションと結合すること。
  - 条件式による条件を満たす場合に、オブザベーションを結合すること。
  - 同一のデータセットからオブザベーションを 2 回読み込むこと。

1 対 1 のマージ

定義

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されたすべてのデータセットから最初のオブザベーションが読み込まれて結合され、新しいデータセットの最初のオブザベーションになります。同様に、2 番目のオブザベーションが読み込まれて結合され、新しいデータセットの 2 番目のオブザベーションになります。以降も同様に処理されます。1 対 1 のマージでは、新しいデータセットに格納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されているデータセットのうち、最も大きなデータセットに含まれているオブザベーションの数と同じになります。

MERGENOBY=システムオプションを使用すると、BY ステートメントを伴わない MERGE ステートメントの処理が発生したときに、メッセージを表示することができます。

構文

SAS データセットをマージするには、MERGE ステートメントを次の形式で使用します。

MERGE *data-set(s)*;

各引数の意味は次のとおりです。

data-set
既存の SAS データセットを 2 つ以上指定します。

注意:
共通する変数の値が重複している場合や異なっている場合は、1 対 1 のマージを実行しないでください。 1 対 1 のマージでは、共通する変数の値が重複しているデータセットの場合、望ましくない出力結果になる可能性があります。ある変数が複数のデータセットに存在する場合、新しいデータセットに書き出される値は、最後に読み込まれるデータセットの値です。変数は、各データセットから読み込まれたものがそのまま結合されます。1 対 1 のマージを使用して、共通する変数の値が異なるデータセットを結合する場合も、望ましくない出力結果になる可能性があります。ある変

数が複数のデータセットに存在する場合、最後に読み込まれるデータセットからの値が欠損値であっても、新しいデータセットに値が書き出されます。あるデータセットのすべてのオブザベーションが処理された後、そのデータセットよりもオブザベーションの数が多いデータセットがある場合、新しいデータセットの以後すべてのオブザベーションで、少ない数のオブザベーションを持つデータセット固有の変数には欠損値が格納されます。

MERGE ステートメントの詳細については、*SAS Statements: Reference* にある同ステートメントの説明を参照してください。

1 対 1 のマージ時の DATA ステップ処理

コンパイルフェーズ

MERGE ステートメント内に指定された各データセットのディスクリプタ情報が読み込まれます。次に、プログラムデータベクトル(PDV)が作成されます。この PDV には、DATA ステップによって作成された変数とともに、すべてのデータセットから読み込まれた変数が格納されます。

実行: ステップ 1

各データセットの最初の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ます。データセットの読み込みは、MERGE ステートメント内に指定したデータセットの順序で行われます。2 つのデータセットに同じ変数が含まれている場合は、2 番目のデータセットから読み込まれた値によって、最初のデータセットからの値が置き換えられます。最後のデータセットの最初のオブザベーションが読み込まれ、DATA ステップ内の他のステートメントがすべて実行されると、プログラムデータベクトルの内容が新しいデータセットに書き出されます。DATA ステップの処理中に作成された変数、または値が割り当てられた変数には、状況によって欠損値が設定される場合があります。

実行: ステップ 2

すべてのデータセットからオブザベーションが読み込まれるまで、読み込みが続けられます。

例 1:1 対 1 のマージ: オブザベーションの数が等しい場合

SAS データセット Animal1 と Plant1 は、どちらも変数 Common を保持しており、オブザベーションは Common の値に基づいて配置されています。入力データセット Animal と Plant の内容は次のとおりです。

```
      Animal                    Plant

OBS  Common  Animal       OBS   Common  Plant

 1     a     Ant           1      a     Apple
 2     b     Bird          2      b     Banana
 3     c     Cat           3      c     Coconut
 4     d     Dog           4      d     Dewberry
 5     e     Eagle         5      e     Eggplant
 6     f     Frog          6      g     Fig
```

次のプログラムは、これらのデータセットを結合した結果を出力します。

```
data combined;
    merge animal plant;
run;

proc print data=combined;
    title 'Data Set Combined';
run;
```

アウトプット 21.7 マージで結合されたデータセット(オブザベーションの数が等しい場合)

Data Set COMBINED

Obs	Common	Animal	Plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	Dewberry
5	e	Eagle	Eggplant
6	g	Frog	Fig

新しいデータセットの各オブザベーションには、入力データセットのすべての変数が格納されます。2 つのデータセットに同じ変数が含まれている場合は、オブザベーション 6 に示されているように、2 番目のデータセットから読み込まれた値によって、最初のデータセットからの値が置き換えられます。

例 2:1 対 1 のマージ: オブザベーションの数が等しくない場合

SAS データセット Animal1 と Plant1 は、どちらも変数 Common を保持しており、オブザベーションは Common の値に基づいて配置されています。データセット Plant1 が持つオブザベーションの数は、データセット Animal1 が持つオブザベーションの数より少なくなっています。入力データセット Animal1 と Plant1 の内容は次のとおりです。

```
      Animal1                    Plant1

OBS   Common   Animal       OBS   Common   Plant

 1      a      Ant           1      a      Apple
 2      b      Bird          2      b      Banana
 3      c      Cat           3      c      Coconut
 4      d      Dog
 5      e      Eagle
 6      f      Frog
```

次のプログラムは、オブザベーションの数が等しくないデータセットを結合した結果を出力します。

```
data combined1;
    merge animal1 plant1;
run;

proc print data=combined1;
    title 'Data Set Combined1';
run;
```

アウトプット 21.8 マージで結合されたデータセット(オブザベーションの数が等しくない場合)

Data Set COMBINED1

Obs	Common	Animal	Plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	
5	e	Eagle	
6	f	Frog	

オブザベーション 4 - 6 の変数 Plant の値が欠損値になっています。

例 3:1 対 1 のマージ: 共通する変数の値が重複している場合

次の例は、共通する変数の値が重複しているデータセットを使用して 1 対 1 のマージを実行した場合に得られる、望ましくない出力結果を示しています。新しいデータセットに書き出されているのは、最後に読み込まれたデータセットの値です。変数は、各データセットから読み込まれたものがそのまま結合されます。次の例では、データセット Animal1 とデータセット Plant1 が変数 Common を保持し、それぞれのデータセットには Common の値が重複しているオブザベーションがあります。入力データセット Animal1 と Plant1 の内容は次のとおりです。

```
      Animal1                    Plant1

OBS   Common   Animal       OBS   Common   Plant

 1      a      Ant           1      a      Apple
 2      a      Ape           2      b      Banana
 3      b      Bird          3      c      Coconut
 4      c      Cat           4      c      Celery
 5      d      Dog           5      d      Dewberry
 6      e      Eagle         6      e      Eggplant
```

次のプログラムは、結合されたデータセット MERGE1 を作成した結果を出力します。

```
   /* This program illustrates undesirable results. */
data merge1;
   merge animal1 plant1;
run;

proc print data=merge1;
   title 'Data Set MERGE1';
run;
```

アウトプット 21.9 望ましくない出力結果(共通する変数の値が重複している場合)

Data Set MERGE1

Obs	Common	Animal1	Plant1
1	a	Ant	Apple
2	b	Ape	Banana
3	c	Bird	Coconut
4	c	Cat	Celery
5	d	Dog	Dewberry
6	e	Eagle	Eggplant

新しいデータセットのオブザベーションの数は 6 つです。オブザベーション 2 とオブザベーション 3 に、望ましくない値が格納されていることに注意してください。ここでは、データセット Animal1 の 2 番目のオブザベーションを読み込みます。次に、データセット Plant1 の 2 番目のオブザベーションを読み込んで、変数 Common と変数 Plant1 の値を置き換えます。3 番目のオブザベーションも同様にして作成します。

例 4:1 対 1 のマージ: 共通する変数の値が異なる場合

次の例は、共通する変数の値が異なるデータセットを使用して、1 対 1 のマージを実行した場合に得られる、望ましくない出力結果を示しています。ある変数が複数のデータセットに存在する場合、最後に読み込まれるデータセットからの値が欠損値であっても、新しいデータセットに値が書き出されます。あるデータセットのすべてのオブザベーションが処理された後、そのデータセットよりもオブザベーションの数が多いデータセットがある場合、新しいデータセットの以後すべてのオブザベーションで、オブザベーションの数が少ないデータセット固有の変数には、欠損値が格納されます。この例では、データセット Animal2 とデータセット Plant2 が持つ変数 Common の値が異なっています。入力データセット Animal2 と Plant2 の内容は次のとおりです。

```
      Animal2                    Plant2

OBS   Common   Animal       OBS    Common   Plant

 1      a      Ant           1       a      Apple
 2      c      Cat           2       b      Banana
 3      d      Dog           3       c      Coconut
 4      e      Eagle         4       e      Eggplant
                             5       f      Fig
```

次のプログラムは、データセット MERGE2 を作成した結果を出力します。

```
   /* This program illustrates undesirable results. */
data merge2;
   merge animal2 plant2;
run;

proc print data=merge2;
   title 'Data Set MERGE2';
run;
```

アウトプット 21.10 望ましくない出力結果(共通する変数の値が異なる場合)

Data Set MERGE2

Obs	Common	Animal2	Plant2
1	a	Ant	Apple
2	b	Cat	Banana
3	c	Dog	Coconut
4	e	Eagle	Eggplant
5	f		Fig

コメントと比較

1 対 1 のマージの結果は、複数の SET ステートメントを使用してオブザベーションを結合する場合に得られる結果と似ています。ただし、1 対 1 のマージでは、MERGE ステートメント内に指定されたデータセットに含まれる、すべてのオブザベーションが処理されます。

マッチマージ

定義

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、BY ステートメントを指定して MERGE ステートメントを使用します。マッチマージを実行するには、すべてのデータセットを、BY ステートメント内に指定した変数に基づいてあらかじめ並べ替えておく必要があります。または、データセットにインデックスを定義する必要があります。

構文

データセットをマッチマージするには、MERGE ステートメントを次の形式で使用します。

MERGE *data-set(s)*;

BY *variable(s)*;

各引数の意味は次のとおりです。

data-set
オブザベーションの読み込み元となる既存のデータセットを 2 つ以上指定します。

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。これらの変数を、BY 変数と呼びます。

MERGE および BY ステートメントの詳細については、*SAS Statements: Reference* にある各ステートメントの説明を参照してください。

マッチマージ時の DATA ステップ処理

コンパイルフェーズ
MERGE ステートメント内に指定された各データセットのディスクリプタ情報が読み込まれ、プログラムデータベクトル(PDV)が作成されます。この PDV には、DATA ステップによって作成された変数とともに、すべてのデータセットから読み込まれた変数が格納されます。BY ステートメントに指定した変数ごとに、自動変数 FIRST.*variable* および LAST.*variable* が作成されます。

実行: ステップ 1
MERGE ステートメントに指定された各データセットに含まれている、最初のオブザベーションが参照され、新しいデータセットの先頭に配置される BY グループが決定されます。その BY グループの最初のオブザベーションが、各データセットからプログラムデータベクトルに読み込まれます。データセットの読み込みは、MERGE ステートメント内に指定したデータセットの順序で行われます。BY グループにオブザベーションが含まれていない場合、その変数については、プログラムデータベクトルに欠損値が格納されます。

実行: ステップ 2
最後のデータセットの最初のオブザベーションが処理され、DATA ステップ内の他のステートメントがすべて実行されると、プログラムデータベクトルの内容が新しいデータセットに書き出されます。プログラムデータベクトル内の変数の値は、DATA ステップによって作成されたものを除き、すべて保持されます。DATA ステップによって作成された変数の値は欠損値に設定されます。オブザベーションの結合は、最初の BY グループに含まれるオブザベーションが、新しいデータセットにすべて書き出されるまで続けられます。すべてのデータセットから、BY グループ内のすべてのオブザベーションが読み込まれると、プログラムデータベクトル内の変数は欠損値に設定されます。各データセットに含まれている次のオブザベーションが参照され、新しいデータセットの 2 番目に配置される BY グループが決定されます。

実行: ステップ 3
データセットにあるすべての BY グループからオブザベーションがすべて読み込まれるまで、これらのステップが繰り返し実行されます。

例 1:条件に基づいてオブザベーションを結合

SAS データセット Animal と Plant がそれぞれ共通の BY 変数 Common を保持し、オブザベーションはその BY 変数の値順に配置されています。入力データセット Animal と Plant の内容は次のとおりです。

```
      Animal                    Plant

OBS  Common  Animal        OBS  Common  Plant

 1     a     Ant           1     a     Apple
 2     b     Bird          2     b     Banana
 3     c     Cat           3     c     Coconut
 4     d     Dog           4     d     Dewberry
 5     e     Eagle         5     e     Eggplant
 6     f     Frog          6     f     Fig
```

次のプログラムは、BY 変数 Common の値に基づいてデータセットを結合した結果を出力します。

```
data combined;
    merge animal plant;
    by Common;
run;

proc print data=combined;
```

```
   title 'Data Set Combined';
run;
```

アウトプット 21.11 *マッチマージによって結合されたデータセット*

Data Set COMBINED

Obs	Common	Animal	Plant
1	a	Ant	Apple
2	b	Bird	Banana
3	c	Cat	Coconut
4	d	Dog	Dewberry
5	e	Eagle	Eggplant
6	f	Frog	Fig

新しいデータセットの各オブザベーションには、入力データセットのすべての変数が格納されます。

例 2: マッチマージ: BY 変数の値が重複している場合

1 つのデータセットの BY グループから最後のオブザベーションが読み込まれると、すべてのデータセットからその BY グループのオブザベーションがすべて読み込まれるまで、データセット固有の変数の値はプログラムデータベクトル内に保持されます。次の例では、データセット Animal1 と Plant1 が持つ BY 変数 Common の値が重複しています。入力データセット Animal1 と Plant1 の内容は次のとおりです。

```
     Animal1                        Plant1

OBS  Common  Animal1          OBS  Common  Plant1

 1     a      Ant              1     a     Apple
 2     a      Ape              2     b     Banana
 3     b      Bird             3     c     Coconut
 4     c      Cat              4     c     Celery
 5     d      Dog              5     d     Dewberry
 6     e      Eagle            6     e
Eggplant
```

次のプログラムは、マージされたデータセット MATCH1 を作成した結果を出力します。

```
data match1;
   merge animal1 plant1;
   by Common;
run;

proc print data=match1;
   title 'Data Set MATCH1';
run;
```

アウトプット 21.12 マッチマージされたデータセット(BY 値が重複している場合)

Data Set MATCH1

Obs	Common	Animal1	Plant1
1	a	Ant	Apple
2	a	Ape	Apple
3	b	Bird	Banana
4	c	Cat	Coconut
5	c	Cat	Celery
6	d	Dog	Dewberry
7	e	Eagle	Eggplant

出力データセットの 2 番目のオブザベーションでは、BY グループに含まれるオブザベーションが新しいデータセットにすべて書き出されるまで、変数 Plant1 の 1 番目のオブザベーションの値が保持され、書き出されます。変数 Animal1 の 4 番目と 5 番目のオブザベーションにおいても同様です。

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。

例 3: マッチマージ: オブザベーションが一致していない場合

入力データセット内にある一致していないオブザベーションに対してマッチマージを実行すると、変数の値は、欠損値であってもすべてプログラムデータベクトル内に保持されます。データセット Animal2 と Plant2 には、BY 変数 Common の値のすべてが格納されてはいません。入力データセット Animal2 と Plant2 の内容は次のとおりです。

```
     Animal2                    Plant2

OBS  Common  Animal2       OBS  Common  Plant2

 1     a      Ant           1     a     Apple
 2     c      Cat           2     b     Banana
 3     d      Dog           3     c     Coconut
 4     e      Eagle         4     e     Eggplant
                            5     f
Fig
```

次のプログラムは、マージされたデータセット MATCH2 を作成した結果を出力します。

```
data match2;
   merge animal2 plant2;
   by Common;
run;
```

```
proc print data=match2;
   title 'Data Set MATCH2';
run;
```

アウトプット 21.13 *マッチマージしたデータセット(オブザベーションが一致していない場合)*

Data Set MATCH2

Obs	Common	Animal2	Plant2
1	a	Ant	Apple
2	b		Banana
3	c	Cat	Coconut
4	d	Dog	
5	e	Eagle	Eggplant
6	f		Fig

出力結果に示されているように、どちらか一方のデータセットにしかない変数内の欠損値も含めて、変数 Common の値はすべて新しいデータセットに格納されます。

UPDATE ステートメントと MODIFY ステートメントによるデータセットの更新

定義

データセットの更新とは、トランザクションデータセットを適用して、マスタデータセットの内容を更新する処理です。データセットを更新するには、2 つの入力データセットを使用します。元の情報を格納しているデータセットをマスタデータセット、新しい情報を格納しているデータセットをトランザクションデータセットといいます。

データセットを更新するには、UPDATE ステートメントまたは MODIFY ステートメントを使用します。

UPDATE ステートメント
: トランザクションデータセットから読み込んだオブザベーションを使用して、マスタデータセット内の対応するオブザベーションの値を更新します。UPDATE ステートメントとともに、BY ステートメントを使用する必要があります。これは、トランザクションデータセット内のオブザベーションは、すべて BY 変数の値をキーとしてマスタデータセット内のオブザベーションと対応付けるためです。

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換、削除、既存データセットへのオブザベーションの追加ができます。MODIFY ステートメントを使用すると、データセットのコピーは作成されず、データセットが直接変更されるので、ディスク領域を節約できます。

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、マスタデータセットに含まれるオブザベーションの数と、トランザクションデータセットに含まれる不一致なオブザベーションの数の合計になります。

UPDATE および MODIFY ステートメントに関する詳細については、*SAS Statements: Reference* にある各ステートメントの説明を参照してください。

UPDATE ステートメントの構文

マスタデータセットを更新するには、UPDATE ステートメントを次の形式で使用します。

UPDATE *master-data-set transaction-data-set*;

BY *variable-list*;

各引数の意味は次のとおりです。

master-data-set
マスタデータセットとして、変更対象の SAS データセットを指定します。

transaction-data-set
トランザクションデータとして、マスタデータセットを更新する SAS データセットを指定します。

variable-list
オブザベーションの対応付けの基準となる BY 変数を指定します。

トランザクションデータセットの BY 変数の値が重複している場合、オブザベーションにはトランザクションデータが両方とも適用されます。新しいデータセットに書き出されるのは、プログラムデータベクトルに最後に読み込まれた値です。トランザクションデータセットがこの形式になっている場合、UPDATE ステートメントではなく MODIFY ステートメントを使用してください。

注意:
マスタデータセット内では、各オブザベーションについての BY 変数の値が一意である必要があります。 マスタデータセット内に、BY 変数の値が等しい 2 つのオブザベーションがある場合には、最初のオブザベーションが更新され、2 番目のオブザベーションは無視されます。DATA ステップを実行すると、SAS ログに警告メッセージが表示されます。

UPDATE ステートメントに関する詳細については、*SAS Statements: Reference* を参照してください。

MODIFY ステートメントの構文

この後の例では、MODIFY ステートメントを次の形式で使用します。

MODIFY *master-data–set*;

BY *variable-list*;

各引数の意味は次のとおりです。

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。

variable-list
データセットの並べ替え基準となる BY 変数を指定します。

注: MODIFY ステートメントでは、変数の追加など、SAS データセットのディスクリプタ部の変更はサポートしていません。

MODIFY ステートメントに関する詳細については、*SAS Statements: Reference* を参照してください。

UPDATE ステートメントを使用した DATA ステップ処理

コンパイルフェーズ

- UPDATE ステートメント内に指定された各データセットのディスクリプタ情報が読み込まれ、プログラムデータベクトル(PDV)が作成されます。この PDV には、DATA ステップによって作成された変数とともに、データセットから読み込まれたすべての変数が格納されます。

- BY ステートメントに指定した変数ごとに、自動変数 FIRST.*variable* および LAST.*variable* が作成されます。

実行: ステップ 1
UPDATE ステートメントに指定された各データセットに含まれている、最初のオブザベーションが参照され、先頭の BY グループが決定されます。トランザクションデータセットの BY 値がマスタデータセットの BY 値よりも先行している場合、読み込みはトランザクションデータセットからのみ行われ、マスタデータセットからの変数は欠損値に設定されます。マスタデータセットの BY 値がトランザクションデータセットの BY 値よりも先行している場合、読み込みはマスタデータセットからのみ行われ、トランザクションデータセットからの変数は欠損値に設定されます。マスタデータセットとトランザクションデータセットの BY 値が等しい場合は、最初の更新処理が適用されます。データセットの最初のオブザベーションの(欠損値でない)値が、プログラムデータベクトルに読み込まれます。

実行: ステップ 2
最初の更新処理が完了すると、トランザクションデータセット内の次のオブザベーションを調べます。BY 値の等しいオブザベーションが存在する場合は、それにも更新処理が適用されます。出力データセットの最初のオブザベーションには、その BY 値における更新処理のうち、最後に行われた更新処理の値が含まれることになります。このオブザベーションのための更新処理が他にない場合は、オブザベーションが新しいデータセットに書き出され、プログラムデータベクトル内の値は欠損値に設定されます。両方のデータセットにあるすべての BY グループからオブザベーションがすべて読み込まれるまで、これらのステップが繰り返し実行されます。

一致しないオブザベーション、欠損値、新しい変数の更新

UPDATE ステートメントでは、マスタデータセット内に存在するオブザベーションに対応するオブザベーションがトランザクションデータセットに存在しない場合、そのようなオブザベーションは変更されずに新しいデータセットに書き出されます。トランザクションデータセットのオブザベーションのうち、マスタデータセットのオブザベーションと対応付けられていないものは、すべてプログラムデータベクトルに書き出され、新しいデータセットのオブザベーションの基礎になります。プログラムデータベクトル内のデータは、新しいデータセットに書き出されるまでは、他の更新処理によって変更される可能性があります。マスタデータセットのオブザベーションを更新する必要がない場合は、対応するオブザベーションをトランザクションデータセットから削除できます。

マスタデータセット内の既存の値に対して、トランザクションデータセット内でピリオド(数値変数の場合)またはブランク(文字変数の場合)を指定しても、欠損値で置き換えられることはありません。欠損値で置き換えるには、欠損値を特殊欠損値の文字で指定したトランザクションデータセットを作成するか、UPDATE ステートメントの UPDATEMODE=NOMISSINGCHECK オプションを使用する必要があります。

UPDATE ステートメントを使用すると、トランザクションデータセット内に、マスタデータセットのすべてのオブザベーションに追加する新しい変数を含めることができます。

これらを使用したサンプルプログラムについては、“例 3:UPDATE ステートメント: 一致しないオブザベーション、欠損値、新しい変数を処理する場合” (492 ページ)を参照してください。

UPDATE ステートメントでは並べ替えが必要

インデックスを使用しない場合は、1 つ以上の同一変数に基づいてマスタデータセットとトランザクションデータセットをあらかじめ並べ替え処理する必要があります。使用する変数は、UPDATE ステートメントとともに指定する BY ステートメントで指定します。マスタデータセット内では、各オブザベーションにおいて BY 変数の値が一意である必要があります。BY 変数を複数使用する場合、マスタデータセットの各オブザベーション内では、すべての BY 変数の値の組み合わせが一意である必要があります。この BY 変数は、更新が不要なものでなければなりません。

注: MODIFY ステートメントの場合、並べ替え処理は必要ありません。ただし、並べ替えることでデータの処理効率が向上します。

MODIFY ステートメントでのインデックスの使用

MODIFY ステートメントはインデックスを維持します。UPDATE ステートメントの場合とは異なり、インデックスを再構築する必要はありません。

UPDATE ステートメントと BY 変数を伴う MODIFY ステートメントの比較

トランザクションをデータセットに適用する方法として、UPDATE ステートメントと、BY 変数を伴う MODIFY ステートメントとを比較します。MODIFY ステートメントには、さまざまな利用方法があります。しかし、UPDATE ステートメントを使用する方が、状況によっては効率の良い場合があります。次の表に、UPDATE ステートメントと BY 変数を伴う MODIFY ステートメントのどちらを使用するかを選択する際のヒントとなる情報を示します。

表 21.3 *BY 変数を伴う MODIFY ステートメントと UPDATE ステートメントとの比較*

観点	BY 変数を伴う MODIFY ステートメント	UPDATE ステートメント
ディスク領域	データを直接更新するため、ディスク領域を節約できます。	データセットの更新済みコピーが作成されるため、必要なディスク領域が多くなります。
並べ替えとインデックス	入力データセットの並べ替えは必須ではありません。ただし、パフォーマンス向上のためには両方のデータセットを並べ替え、マスタデータセットのインデックスを作成することが推奨されます。	入力データセットを、あらかじめ並べ替えることが必要です。
使用するタイミング	データセットのごく一部を更新する場合に使用します。	データセットの大部分を更新する必要がある場合に使用します。
変更済みデータセットを指定する位置	更新済みデータセットを DATA ステートメントと MODIFY ステートメントの両方に指定します。	更新済みデータセットを DATA ステートメントと UPDATE ステートメントに指定します。
重複する BY 値	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重複する BY 値があっても更新処理できます。	トランザクションデータセットのみ、重複する BY 値があっても更新処理できます。マスタデータセットに重複する BY 値がある場合は、警告メッセージを表示します。
有効範囲	データセットのディスクリプタ情報を変更できないため、変数または変数ラベルの追加や削除などは実行できません。	新しい変数の追加など、データセットのディスクリプタ部の情報を変更できます。
エラーチェック	自動変数_IORC_と SYSRC 自動呼び出しマクロを使用してエラーチェックを実行できます。	トランザクションデータセットは、対応するマスタデータセットのオブザベーションが存在しない場合は適用されず、データセットに追加されるため、エラーチェックは不要です。
データセットの整合性	タスクが異常終了した場合、データの一部のみが更新されることがあります。	データセットのコピーを作成して処理するため、データセットは消失しません。

SAS データセットの結合に関する詳細については、表 21.2 (463 ページ)を参照してください。

MODIFY ステートメントの主な用途

MODIFY ステートメントの主な用途には次の 3 つがあります。

- 単一の SAS データセット内にあるオブザベーションを変更します。
- 単一の SAS データセット内にあるオブザベーションを、オブザベーション番号またはインデックス値に基づいて直接変更します。
- トランザクションデータセット内にある値を使用して、マスタデータセット内にあるオブザベーションを変更します。MODIFY ステートメントと BY ステートメントを使用すると、UPDATE ステートメントを使用することとほぼ同じになります。

これらの使用方法は、以降の例で紹介します。

例 1:UPDATE ステートメント: 基本的な更新をする場合

この例では、データセット Master に変数 Animal と変数 Plant の元の値が格納されています。データセット NEWPlant は、変数 Plant の新しい値を保持するトランザクションデータセットです。入力データセット Master および NEWPlant の内容は次のとおりです。

```
      Master                          NEWPlant

OBS Common Animal Plant           OBS Common Plant

 1    a    Ant    Apple            1    a    Apricot
 2    b    Bird   Banana           2    b    Barley
 3    c    Cat    Coconut          3    c    Cactus
 4    d    Dog    Dewberry         4    d    Date
 5    e    Eagle  Eggplant         5    e    Escarole
 6    f    Frog   Fig              6    f    Fennel
```

次のプログラムは、データセット NEWPlant をトランザクションデータに使用してデータセット Master を更新した結果を、UPDATE_FILE に出力します。

```
data update_file;
   update master newplant;
   by common;
run;

proc print data=update_file;
   title 'Data Set Update_File';
run;
```

アウトプット 21.14　トランザクションデータセットにより更新されたマスタデータセット

Data Set Update_File

Obs	Common	Animal	Plant
1	a	Ant	Apricot
2	b	Bird	Barley
3	c	Cat	Cactus
4	d	Dog	Date
5	e	Eagle	Escarole
6	f	Frog	Fennel

更新されたデータセットの各オブザベーションには、変数 Plant の新しい値が格納されます。

例 2:UPDATE ステートメント: BY 変数の値が重複している場合

マスタデータセット内に、BY 変数の値が等しい 2 つのオブザベーションがある場合には、最初のオブザベーションが更新され、2 番目のオブザベーションは無視されます。SAS ログには警告メッセージが表示されます。トランザクションデータセットの BY 変数の値が重複している場合、オブザベーションにはトランザクションデータが両方とも適用されます。新しいデータセットに書き出されるのは、プログラムデータベクトルに最後に読み込まれた値です。入力データセット Master1 と DupPlant の内容は次のとおりです。

```
      Master1                          DupPlant

OBS Common Animal1 Plant1          OBS Common Plant1

 1    a    Ant     Apple            1    a    Apricot
 2    b    Bird    Banana           2    b    Barley
 3    b    Bird    Banana           3    c    Cactus
 4    c    Cat     Coconut          4    d    Date
 5    d    Dog     Dewberry         5    d    Dill
 6    e    Eagle   Eggplant         6    e    Escarole
 7    f    Frog    Fig              7    f    Fennel
```

次のプログラムは、DupPlant をトランザクションデータに使用してデータセット Master1 を更新し、その結果を出力します。

```
data update1;
   update master1 dupplant;
   by Common;
run;

proc print data=update1;
   title 'Data Set Update1';
run;
```

アウトプット 21.15 更新されたデータセット(BY 値が重複している場合)

Data Set Update1

Obs	Common	Animal1	Plant1
1	a	Ant	Apricot
2	b	Bird	Barley
3	b	Bird	Banana
4	c	Cat	Cactus
5	d	Dog	Dill
6	e	Eagle	Escarole
7	f	Frog	Fennel

この DATA ステップを実行すると、BY グループに複数のオブザベーションがあるという警告メッセージが表示されます。しかし、DATA ステップの処理は続行され、データセット Update1 が作成されます。

出力データセットには、7 つのオブザベーションが格納されています。2 番目と 3 番目オブザベーションで、BY 変数 Common の値が重複しています。変数 Plant1 の値は、重複している 2 番目の BY 値では更新されませんでした。

例 3:UPDATE ステートメント: 一致しないオブザベーション、欠損値、新しい変数を処理する場合

この例では、データセット Master2 がマスタデータセットです。このデータセットの最初のオブザベーションには、変数 Plant2 に欠損値が含まれていますが、BY 変数 Common の値のすべては含まれていません。トランザクションデータセット NONPlant には、新しい変数 Mineral、BY 変数 Common の新しい値、およびいくつかのオブザベーションの欠損値が含まれています。入力データセット Master2 と NONPlant の内容は次のとおりです。

Master2

OBS	Common	Animal2	Plant2
1	a	Ant	
2	c	Cat	Coconut
3	d	Dog	Dewberry
4	e	Eagle	Eggplant
5	f	Frog	Fig

NONPlant

OBS	Common	Plant2	Mineral
1	a	Apricot	Amethyst
2	b	Barley	Beryl
3	c	Cactus	
4	e		
5	f	Fennel	
6	g	Grape	Garnet

次のプログラムは、データセット Master2 を更新し、その結果を出力します。

```
data update2_file;
   update master2 nonplant;
   by Common;
run;

proc print data=update2_file;
```

```
   title 'Data Set Update2_File';
run;
```

アウトプット 21.16 *更新されたデータセット(新しい変数、一致しないオブザベーション、欠損値を処理する場合)*

Data Set Update2_File

Obs	Common	Animal2	Plant2	Mineral
1	a	Ant	Apricot	Amethyst
2	b		Barley	Beryl
3	c	Cat	Cactus	
4	d	Dog	Dewberry	
5	e	Eagle	Eggplant	
6	f	Frog	Fennel	
7	g		Grape	Garnet

この結果が示すように、すべてのオブザベーションに変数 Mineral の値が含まれています。いくつかのオブザベーションでは、Mineral の値が欠損値に設定されています。トランザクションデータセットに含まれる 2 番目と 6 番目のオブザベーションに対応するオブザベーションは、Master2 にはなく、これらは新しく追加されたオブザベーションになります。マスタデータセットから読み込まれた 3 番目のオブザベーションは、変更されないまま新しいデータセットに書き出されています。4 番目のオブザベーションにある変数 Plant2 の値は欠損値に変更されていません。新しいデータセットにある残りのオブザベーションには、変数 Plant2 の更新済みの値が保持されています。

次のプログラムは、UPDATE ステートメントの UPDATEMODE=オプションを使用し、その結果を出力します。

```
data update2_file;
   update master2 nonplant updatemode=nomissingcheck;
   by Common;
run;

proc print data=update2_file;
   title 'Data Set Update2_File - UPDATEMODE Option';
run;
```

アウトプット 21.17 *UPDATEMODE オプションによる更新の結果*

Data Set Update2_File - UPDATEMODE Option

Obs	Common	Animal2	Plant2	Mineral
1	a	Ant	Apricot	Amethyst
2	b		Barley	Beryl
3	c	Cat	Cactus	
4	d	Dog	Dewberry	
5	e	Eagle		
6	f	Frog	Fennel	
7	g		Grape	Garnet

5 番目のオブザベーションに含まれる変数 Plant2 の値は欠損値に設定されています。これは、UPDATEMODE=NOMISSINGCHECK オプションが機能しているためです。

データセット更新の詳細な例については、*Combining and Modifying SAS Data Sets:Examples* を参照してください。

例 4:オブザベーションの追加によりマスタデータセットを更新する場合

トランザクションデータセットの中に、マスタデータセット内のオブザベーションと一致していないオブザベーションが含まれている場合は、プログラムを変更する必要があります。次の例では、MODIFY 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 Add_Inventory からの値でマスタデータセット Inventory を更新します。入力データセット Inventory および Add_Inventory の内容は次のとおりです。

Master

Part Number	Description	In Stock	Received Date	Price
K89R	seal	34	27jul1998	245.00
M4J7	sander	98	20jun1998	45.88
LK43	filter	121	19may1999	10.99
MN21	brace	43	10aug1999	27.87
BC85	clamp	80	16aug1999	9.55
NCF3	valve	198	20mar1999	24.50
KJ66	cutter	6	18jun1999	19.77
UYN7	rod	211	09sep1999	11.55
JD03	switch	383	09jan2000	13.99
BV1E	timer	26	03aug2000	34.50

```
          Add_Inventory

Part                    In
Number  Description     Stock  Price

K89R    seal             6     247.50
AA11    hammer          55      32.26
BB22    wrench          21      17.35
KJ66    cutter          10      24.50
CC33    socket           7      22.19
BV1E    timer           30      36.50
```

たとえば、入力データセット Add_Inventory のオブザベーション 2 は、マスタデータセット Inventory に一致するものがありません。したがって、新しいオブザベーションをマスタデータセットに書き出す場合は、OUTPUT ステートメントを明示的に使用する必要があります。[1]

OUTPUT ステートメントを明示的に指定した場合は、REPLACE ステートメントも指定して、両方のデータセットにあるオブザベーションを更新する必要があります。

次のプログラムでは、MODIFY、UPDATE および REPLACE ステートメントを使用して、マスタデータセットに新しいオブザベーションを追加し、既存オブザベーションを更新します。

例のコード 21.1 *Updating a Master Data Set By Adding Observations*

```
data INVENTORY;
   modify Inventory Add_Iventory; 1
   by PartNumber;
   select (_iorc_); 2
     when (%sysrc(_sok)) 3
     do;
       InStock=InStock+NewStock;
       ReceivedDate=today();
       Price=NewPrice;
       replace; 4
     end;
     when (%sysrc(_dsenmr)) 5
     do;
       InStock=NewStock;
       ReceivedDate=today();
       Price=NewPrice;
       output; 6
       _error_=0;
     end;
     otherwise 7
     do;
       put 'An unexpected I/O error has occurred.'
       _error_=0;
       stop;
     end;
   end;
```

1 DATA ステップでの MODIFY ステートメントのデフォルト動作は OUTPUT ではなく REPLACE なので、OUTPUT ステートメントが必要です。

```
run;
proc print data=INVENTORY;
  title 'Data Set INVENTORY (Updated)';
run;
quit;
```

1 MODIFY ステートメントを使用して、データセット Inventory および Add_Inventory からデータをロードします。

2 SELECT ステートメントを使用して、ステートメントのグループを条件に基づいて処理します。

3 SYSRC 自動呼び出しマクロで自動変数_IORC_を使用して、エラーをチェックします。_IORC_の値が_SOK の場合、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データセットのオブザベーションが一致します。

4 REPLACE ステートメントを使用して、データセット Inventory をトランザクションデータセットのオブザベーションで更新します。

5 SYSRC 自動呼び出しマクロで自動変数_IORC_を使用して、エラーをチェックします。_IORC_の値が_DSENMR の場合、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データセットに存在しません。

6 OUTPUT ステートメントを使用して、マスタデータセットの最後に現在のオブザベーションを書き出します。

7 どちらの条件も満たされない場合は、PUT ステートメントを使用して、ログにメッセージを書き出します。

アウトプット 21.18 *オブザベーションの追加によるマスタデータセット更新の結果*

Data Set INVENTORY (Updated)

Obs	PartNumber	Description	InStock	ReceivedDate	Price
1	K89R	seal	40	05FEB2013	247.50
2	M4J7	sander	98	20JUN1998	45.88
3	LK43	filter	121	19MAY1999	10.99
4	MN21	brace	43	10AUG1999	27.87
5	BC85	clamp	80	16AUG1999	9.55
6	NCF3	valve	198	20MAR1999	24.50
7	KJ66	cutter	16	05FEB2013	24.50
8	UYN7	rod	211	09SEP1999	11.55
9	JD03	switch	383	09JAN2000	13.99
10	BV1E	timer	56	05FEB2013	36.50
11	AA11	hammer	55	05FEB2013	32.26
12	BB22	wrench	21	05FEB2013	17.35
13	CC33	socket	7	05FEB2013	22.19

この例では、DATA ステップで、マスタデータセット Inventory がトランザクションデータセット Add_Inventory に基づいて更新されます。プログラムでは、OUTPUT ステートメントを使用して、マスタデータセットに Add_Inventory のオブザベーション 2、3 および 5 を追加します。また、REPLACE ステートメントも使用して、オブザベーション 1、4 およ

び 6 を InStock の新しい値で更新します。_IORC_自動変数がエラーチェックのために使用されます。

_IORC_とエラーチェックの詳細については、“インデックスを用いたデータのランダムなアクセスまたは更新時のエラーチェック” (497 ページ)を参照してください。条件に基づいた SELECT ステートメントの使用については、“SELECT Statement” (*SAS Statements: Reference*)を参照してください。

インデックスを用いたデータのランダムなアクセスまたは更新時のエラーチェック

エラーチェックの重要性

SET ステートメントで KEY=オプション、または MODIFY ステートメントを使用してオブザベーションを読み込む場合は、エラーチェックが非常に重要になります。最も大きな理由は、これらの機能ではシーケンシャルアクセスを使用していないということです。そのため、要求を満たす位置にオブザベーションが格納されるという保証がありません。エラーチェックを行うと、入出力操作の結果に応じて、特定の処理を行うことができます。プログラムは、予想どおりの状況になった場合は実行を続け、予期できない状況が発生した場合は実行を終了します。

エラーチェックツール

MODIFY ステートメントを使用するか、または SET ステートメントで KEY=オプションを使用して SAS データセットを処理する場合に、簡単にエラーチェックを行うためのツールが 2 つ用意されています。

- _IORC_自動変数
- SYSRC 自動呼び出しマクロ

自動変数_IORC_は、MODIFY ステートメント、または SET ステートメントで KEY=オプションを使用すると自動的に作成されます。自動変数_IORC_の値は、最後に実行された MODIFY ステートメント、または最後に実行された KEY=オプションを伴う SET ステートメントによる、入出力操作のステータスを示す数値のリターンコードです。この変数の値を調べることにより、アプリケーションを終了させることなく、異常な入出力ステータスを検出し、特定の処理を行うことができます。たとえば、KEY=変数値が 2 つのオブザベーションの間で一致する場合には、それらを結合してオブザベーションを出力することができます。一致しない場合に SAS ログへメッセージを書き出すのみにすることもできます。

自動変数_IORC_の値は内部処理で使用されるため、変更されやすくなっています。そのため、_IORC_値の変更の影響を受けないで、特定の入出力の状態を調べることができるようにするために、SYSRC マクロが開発されました。SYSRC マクロを使用するときは、次の表に示したニーモニックのいずれかを指定することで、自動変数_IORC_の値を調べることができます。

表 21.4 *DATA ステップ処理用の_IORC_の代表的なニーモニック値*

ニーモニック値	意味	発生条件
_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データセットに存在しません。	BY 変数を伴う MODIFY ステートメントが使用され、一致するデータが存在しない場合。

ニーモニック値	意味	発生条件
_DSEMTR	同じ BY 変数値を持つ、トランザクションデータセットの複数のオブザベーションがマスタデータセットに存在しません。	BY 変数を伴う MODIFY ステートメントが使用され、最初のデータセット内に、同じ BY 値を持つ連続するオブザベーションと一致するオブザベーションが見つからない場合。一致するオブザベーションが見つからなかった場合、最初のオブザベーションは、_DSENMR_を返します。後続のオブザベーションは_DSEMTR_を返します。
_DSENOM	一致するオブザベーションがマスタデータセットに存在しません。	SET ステートメント、または MODIFY ステートメントで KEY=オプションが使用され、一致するオブザベーションが存在しない場合。
_SENOCHN	出力操作が失敗しました。	MODIFY ステートメントで KEY=オプションに重複する値が含まれている場合。
_SOK	入出力操作が成功しました。	一致するオブザベーションが見つかった場合。

例 1: 予期しない状況が発生した場合の実行手順

概要

この例では、予期できない状況が発生した場合に DATA ステップが終了するのを防ぐ方法を示します。処理の目的は、トランザクションデータセットから取得した新しいデータを使用してマスタデータセットを更新することです。共通する変数の値がどちらのデータセットでも重複していないことが、前提条件になります。

注: このプログラムが期待したとおりに動作するのは、マスタデータセットにもトランザクションデータセットにも共通する変数において、オブザベーションに重複する値が存在しない場合だけです。重複する値が存在する場合、KEY=オプションを伴う MODIFY ステートメントがどのように動作するかについては、*SAS Statements: Reference* にある MODIFY ステートメントの説明を参照してください。

入力データセット

データセット Transaction には、3 つのオブザベーションが格納されています。格納されているオブザベーションは、データセット Master を更新するための 2 つのオブザベーションと、変数 PartNumber の値が 6 である新しいオブザベーション(追加する必要があるもの)です。データセット Master は、PartNumber でインデックス付けされています。データセット Master と Transaction の PartNumber の値は、いずれも重複していません。入力データセット Master と Transaction の内容は次のとおりです。

```
          Master                          Transaction

OBS    PartNumber    Quantity       OBS    PartNumber    AddQuantity

 1         1           10            1         4            14
```

```
2        2          20          2         6          16
3        3          30          3         2          12
4        4          40
5        5          50
```

変更前のプログラム

このプログラムの目的は、データセット Transaction の情報を使用してデータセット Master を更新することです。データセット Transaction はシーケンシャルに読み込まれます。データセット Master は、シーケンシャルではなく、MODIFY ステートメントで KEY=オプションを使用して直接読み込まれます。データセット Master から読み込まれるのは、KEY=変数である PartNumber の値が一致しているオブザベーションのみです。

```
data master;  1
   set transaction;  2
   modify master key=PartNumber;  3
   Quantity = Quantity + AddQuantity;  4
run;
```

1 データセット Master を更新用に開きます。

2 データセット Transaction からオブザベーションを読み込みます。

3 データセット Master から読み込んだオブザベーションを、変数 PartNumber の値に基づいて対応付けます。

4 データセット Transaction から読み込んだ新しい値を追加することで、変数 Quantity の情報を更新します。

結果ログ

1 つのオブザベーションは正しく更新されましたが、変数 PartNumber の値が 6 と一致するオブザベーションを見つけられなかったため、このプログラムは停止しました。次の行が SAS ログに書き出されます。

```
ERROR: No matching observation was found in Master data set.
PartNumber=6 AddQuantity=16 Quantity=70 _ERROR_=1
_IORC_=1230015 _N_=2
NOTE: The SAS System stopped processing this step because
      of errors.
NOTE: The data set WORK.MASTER has been updated.  There were
      1 observations rewritten, 0 observations added and 0
      observations deleted.
```

結果データセット

データセット Master は正しく更新されませんでした。更新済みのマスタデータセットには 5 つのオブザベーションがあります。1 つのオブザベーションは正しく更新されましたが、新しいオブザベーションは追加されておらず、2 番目の更新は実行されていません。正しく更新されなかったデータセット Master の内容は次のとおりです。

```
          Master

OBS    PartNumber    Quantity
 1         1            10
 2         2            20
 3         3            30
 4         4            54
 5         5            50
```

変更後のプログラム

このプログラムの目的は、Master に 2 つの更新と 1 つの追加を適用することです。この操作により、データセット Transaction の変数 PartNumber の値 6 と一致するオブザベーションがデータセット Master にない場合でも、DATA ステップは停止することなく処理を行います。エラーチェックを追加することで、DATA ステップは正常に処理を完了し、正しく変更された Master を作成できます。このプログラムでは、SELECT グループ内で自動変数_IORC_と SYSRC 自動呼び出しマクロを使用して_IORC_の値をチェックします。一致するオブザベーションが見つかると、適切なコードを実行します。

```
data master;  1
   set transaction;  2
   modify master key=PartNumber;  3

select(_iorc_);  4
      when(%sysrc(_sok)) do;
         Quantity = Quantity + AddQuantity;
         replace;
      end;
      when(%sysrc(_dsenom)) do;
         Quantity = AddQuantity;
         _error_ = 0;
         output;
      end;
      otherwise do;
         put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
         put 'Program terminating. DATA step iteration # ' _n_;
         put _all_;
         stop;
      end;
   end;
run;
```

1 データセット Master を更新用に開きます。

2 データセット Transaction からオブザベーションを読み込みます。

3 データセット Master から読み込んだオブザベーションを、変数 PartNumber の値に基づいて対応付けます。

4 Master の中に PartNumber 値に一致するオブザベーションが見つかったかどうかに基づいて、適切な処理を実行します。Transaction から読み込んだ新しい値を追加して、Quantity を更新します。SELECT グループによって、正しい処理が行われます。一致するオブザベーションが見つかった(ニーモニックが_SOK)場合、Quantity を更新し、Master にある元のオブザベーションを置き換えます。一致するオブザベーションがない(ニーモニックが_DSENOM)場合、Quantity を Transaction から読み込んだ AddQuantity と同じ値に設定し、新しいオブザベーションを追加します。自動変数_ERROR_は 0 に再設定されます。これは、エラー状態によりプログラムデータベクトルの内容が SAS ログに書き出されないようにするためです。予期できない状況が発生すると、メッセージとプログラムデータベクトルの内容が SAS ログに書き出され、DATA ステップは停止します。

結果ログ

この DATA ステップは、エラーもなく実行され、オブザベーションは適切に更新および追加されます。次の行が SAS ログに書き出されます。

```
NOTE: The data set WORK.MASTER has been updated.  There were
      2 observations rewritten, 1 observations added and 0
      observations deleted.
```

正しく更新された Master データセット

データセット Master には、変数 PartNumber の値が 2 と 4 について更新された変数 Quantity、および変数 PartNumber の値 6 について新しいオブザベーションが格納されています。正しく更新されたデータセット Master の内容は次のとおりです。

Master

OBS	PartNumber	Quantity
1	1	10
2	2	32
3	3	30
4	4	54
5	5	50
6	6	16

例 2:KEY=オプションを使用したステートメントに対するエラーチェック

概要

この例では、データ読み込み時に KEY=オプションを使用して、ステートメントすべてを対象としてエラーチェックを実施することの重要性を示します。

入力データセット

データセット Master とデータセット Description は、どちらも変数 PartNumber でインデックス付けされています。データセット Order には、1 つの注文に含まれるすべての部品の値が格納されています。データセット Order のみが変数 PartNumber の値で 8 を持っています。Master、Order、Description の各入力データセットの内容は次のとおりです。

Master

OBS	PartNumber	Quantity
1	1	10
2	2	20
3	3	30
4	4	40
5	5	50

ORDER

OBS	PartNumber
1	2
2	4
3	1
4	3
5	8
6	5
7	6

Description

OBS	PartNumber	PartDescription
1	4	Nuts
2	3	Bolts
3	2	Screws
4	6	Washers

論理エラーのある変更前のプログラム

このプログラムの目的は、1 つの注文に含まれる部品それぞれについての説明と在庫数を保持するデータセットを作成することです。ただし、Master と Description という 2 つの入力データセットのどちらにも含まれていない部品は対象外とします。データセット Order には、1 つの注文に含まれるすべての部品の部品番号が格納されていま

す。データセット Description は部品の説明を取り出すために読み込まれ、データセット Master は在庫数量を取り出すために読み込まれます。

このプログラムは、データセット Order をシーケンシャルに読み込みます。次に KEY= オプションを指定した SET ステートメントを使用して、データセット Master と Description を直接読み込みます。この読み込みは、PartNumber のキー値に基づいています。一致するオブザベーションが見つかると、Order の各変数 PartNumber の値について必要な情報がすべて含まれたオブザベーションを書き出します。データセットを読み込むために KEY=オプションを指定した 2 つの SET ステートメントのうち、1 つのステートメントでエラーチェックを利用しています。

```
data combine;  1
   length PartDescription $ 15;
   set order;  2
   set description key=PartNumber;  2
   set master key=PartNumber;  2
   select(_iorc_);  3
         when(%sysrc(_sok)) do;
            output;
         end;
         when(%sysrc(_dsenom)) do;
            PartDescription = 'No description';
            _error_ = 0;
            output;
         end;
         otherwise do;
            put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
            put 'Program terminating.';
            put _all_;
            stop;
         end;
   end;
run;
```

1 データセット Combine を作成します。

2 データセット Order からオブザベーションを読み込みます。キー変数である PartNumber の値が一致しているオブザベーションについて、データセット Description とデータセット Master からオブザベーションを読み込みます。Description からオブザベーションを読み込んだ後は、エラーチェックが実行されないことに注意が必要です。

3 Master または Description の中に変数 PartNumber の一致するオブザベーションが見つかったかどうかに基づいて、適切な処理を実行します。この SELECT グループのロジックは、先行する KEY=オプションを使用する 2 つの SET ステートメントの両方に対してエラーチェックが実行されるという、誤った仮定に基づいています。実際には、最後に実行されたステートメントに対してのみエラーチェックが実行されます。SELECT グループによって、正しい処理が行われます。一致するオブザベーションが見つかった(ニーモニックが _SOK)場合、Master から読み込まれるオブザベーションにある PartNumber の値が、Order から読み込まれた現在の PartNumber の値と一致しています。したがって、オブザベーションが出力されます。一致するオブザベーションがない(ニーモニックが _DSENOM)場合、Master から読み込まれるオブザベーションには PartNumber の現在の値が含まれていません。したがって、PartDescription の値が適切に設定され、オブザベーションが出力されます。自動変数_ERROR_は 0 に再設定されます。これは、エラー状態によりプログラムデータベクトルの内容が SAS ログに書き出されないようにするためです。予期できない状況が発生すると、メッセージとプログラムデータベクトルの内容が SAS ログに書き出され、DATA ステップは停止します。

結果ログ

このプログラムを実行すると出力データセットが作成されますが、エラーが 1 つ発生します。次の行が SAS ログに書き出されます。

```
PartNumber=1 PartDescription=Nuts Quantity=10 _ERROR_=1
_IORC_=0 _N_=3
PartNumber=5 PartDescription=No description Quantity=50
_ERROR_=1 _IORC_=0 _N_=6
NOTE: The data set WORK.COMBINE has 7 observations and 3 variables.
```

結果データセット

次の図は、正しく作成されなかったデータセット Combine を示しています。5 番目のオブザベーションの値は不適切な値を示しています。変数 PartNumber の値 8 は、Master と Description のどちらにも Quantity の値が格納されていないのに、表示されるのは不適切な状態ということになります。また、3 番目のオブザベーションと 7 番目のオブザベーションには、それぞれ 2 番目のオブザベーションと 6 番目のオブザベーションの説明が格納されています。

Combine

OBS	PartNumber	PartDescription	Quantity
1	2	Screws	20
2	4	Nuts	40
3	1	Nuts	10
4	3	Bolts	30
5	8	No description	30
6	5	No description	50
7	6	No description	50

変更後のプログラム

この例では、正確な出力データセットを作成するために、KEY=オプションを使用する 2 つの SET ステートメントの両方に対してエラーチェックを実行しています。

```
data combine(drop=Foundes);  1
   length PartDescription $ 15;
   set order; 2
   Foundes = 0;  3
   set description key=PartNumber;  4
   select(_iorc_);  5
      when(%sysrc(_sok)) do;
         Foundes = 1;
      end;
      when(%sysrc(_dsenom)) do;
         PartDescription = 'No description';
         _error_ = 0;
      end;
      otherwise do;
         put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
         put 'Program terminating. Data set accessed is Description';
         put _all_;
         _error_ = 0;
         stop;
      end;
   end;
set master key=PartNumber;  6
select(_iorc_); 7
```

```
      when(%sysrc(_sok)) do;
         output;
      end;
      when(%sysrc(_dsenom)) do;
         if not Foundes then do;
            _error_ = 0;
            put 'WARNING: PartNumber ' PartNumber 'is not in'
                ' Description or Master.';
         end;
         else do;
            Quantity = 0;
            _error_ = 0;
            output;
         end;
      end;
      otherwise do;
         put 'ERROR: Unexpected value for _IORC_= ' _iorc_;
         put 'Program terminating. Data set accessed is Master';
         put _all_;
         _error_ = 0;
         stop;
      end;
   end;      /* ends the SELECT group */
run;
```

1 データセット Combine を作成します。

2 データセット Order からオブザベーションを読み込みます。

3 変数 Foundes を作成します。この変数の値は、データセット Description 内で変数 PartNumber の値と一致するオブザベーションが見つかった箇所を示すフラグとして、後ほど使用します。

4 変数 PartNumber をキー変数として使用し、データセット Description からオブザベーションを読み込みます。

5 Description の中に変数 PartNumber の値と一致するオブザベーションが見つかったかどうかに基づいて、適切な処理を実行します。自動変数_IORC_の値に基づき、SELECT グループによって正しい処理が行われます。一致するオブザベーションが見つかった(ニーモニックが_SOK)場合は、Description から読み込まれたオブザベーションにある変数 PartNumber の値が、Order から読み込まれた現在の PartNumber の値と一致しています。Foundes が 1 に設定されます。これは、Description からの値が現在のオブザベーションに反映されていることを示します。一致するオブザベーションがない(ニーモニックが_DSENOM)場合は、Description から読み込まれるオブザベーションには PartNumber の現在の値が含まれていません。したがって、変数 PartDescription には No description が割り当てられます。自動変数_ERROR_は 0 に再設定されます。これは、エラー状態によりプログラムデータベクトルの内容が SAS ログに書き出されないようにするためです。これ以外の自動変数_IORC_の値は、すべて予期できない状況が発生したことを示します。そのため、SAS ログにメッセージが表示され、DATA ステップは停止します。

6 PartNumber をキー変数として使用し、データセット Master からオブザベーションを読み込みます。

7 Master の中に PartNumber 値に一致するオブザベーションが見つかったかどうかに基づいて、適切な処理を実行します。Order と Master から読み込んだ現在の PartNumber の値に一致するオブザベーションが見つかった(ニーモニックが_SOK)場合は、オブザベーションを書き出します。Master の中に一致するオブザベーションが見つからなかった(ニーモニックが_DSENOM)場合は、Foundes の値を調べま

す。Foundes が真でない場合は、Description の中にも値が見つかりません。そのため、SAS ログにメッセージを表示します。オブザベーションは書き出しません。Foundes が真の場合は、Description の中には値があるものの、Master の中にはありません。そのため、オブザベーションは書き出しますが、Quantity は 0 に設定します。ここでも、予期できない状況が発生した場合は SAS ログにメッセージを表示して、DATA ステップを停止します。

結果ログ

DATA ステップを実行してもエラーは発生しません。6 つのオブザベーションが正しく作成され、SAS ログに次のメッセージが表示されます。

```
WARNING: PartNumber 8 is not in Description or Master.
NOTE: The data set WORK.COMBINE has 6 observations
      and 3 variables.
```

正しく作成されたデータセット Combine

正しく作成されたデータセット Combine を次に示します。Combine の中に、変数 PartNumber の値 8 を持つオブザベーションが含まれなくなりました。この値は Master にも Description にも格納されていない値のため、データセットは正しく作成されました。

Combine

OBS	PartNumber	PartDescription	Quantity
1	2	Screws	20
2	4	Nuts	40
3	1	No description	10
4	3	Bolts	30
5	5	No description	50
6	6	Washers	0

22 章

DATA ステップコンポーネントオブジェクトの使用

DATA ステップコンポーネントオブジェクトの概要

DATA ステップで使用できる次の 5 つの事前定義されたコンポーネントオブジェクトが提供されています。

ハッシュオブジェクトとハッシュ反復子オブジェクト
ルックアップキーに基づいたデータの保存、検索、取り出しが迅速かつ効率的に行えるようにします。ハッシュオブジェクトのキーおよびデータは、DATA ステップ変数になります。キーおよびデータの値は、直接割り当てられた定数値か、または SAS データセットから読み込まれた値のいずれかになります。言語要素としてのハッシュオブジェクトとハッシュ反復子オブジェクトの詳細については、“Dictionary of Hash and Hash Iterator Object Language Elements” (*SAS Component Objects: Reference*)を参照してください。

Java オブジェクト
Java Native Interface (JNI)と同様のメカニズムを提供することにより、Java クラスのインスタンス作成や、結果として得られるオブジェクトのフィールドやメソッドへのアクセスを可能にします。詳細については、“Dictionary of Java Object Language Elements” (*SAS Component Objects: Reference*)を参照してください。

ロガーオブジェクトとアペンダオブジェクト
ログイベントを記録し、それらのイベントを適切な出力先へと書き出せるようにします。詳細については、“Component Object Reference” (*SAS Logging: Configuration and Programming Reference*)を参照してください。

DATA ステップコンポーネントインターフェイスは、ユーザーがステートメント、属性、演算子、メソッドを使用して、コンポーネントオブジェクトを作成および操作することを可能にします。DATA ステップコンポーネントオブジェクトの属性やメソッドにアクセスするには、DATA ステップオブジェクトのドット表記を使用します。ドット表記や、DATA ステップオブジェクトのステートメント、属性、メソッド、演算子の詳細については、*SAS コンポーネントオブジェクト: リファレンス*にある Dictionary of Component Language Elements を参照してください。

注: DATA ステップコンポーネントオブジェクトのステートメント、属性、メソッド、演算子は、各オブジェクトで定義されているものに限定されます。これらの事前定義されている DATA ステップオブジェクトでは、SAS コンポーネント言語(SCL)機能を使用できません。

ハッシュオブジェクトの使用

ハッシュオブジェクトを使用する理由

ハッシュオブジェクトは、データの保存や取得をすばやく行うための効率的なメカニズムを提供します。ハッシュオブジェクトは、ルックアップキーに基づいてデータを保存および取得します。

DATA ステップコンポーネントオブジェクトインターフェイスを使用するには、次の操作を実行します。

1. ハッシュオブジェクトを宣言します。
2. ハッシュオブジェクトのインスタンスを作成します。
3. ルックアップキーとデータを初期化します。

ハッシュオブジェクトを宣言し初期化すると、以下を含む多くのタスクを実行できるようになります。

- データの保存と取り出し
- キーサマリーの管理
- データの置換と削除

- ハッシュオブジェクトの比較
- ハッシュオブジェクト内のデータを含むデータセットの書き出し

たとえば、固有の患者番号と体重に対応する数値結果を含む大きなデータセットがあるとします。また、この大きなデータセットのサブセットである患者番号を含む小さなデータセットもあるとします。この場合、固有の患者番号をキーとし、体重をデータとして使用して、この大きなデータセットをハッシュオブジェクトへロードすることができます。その後、患者番号を使用してハッシュオブジェクト内の現在の患者の検索を繰り返すことにより、患者の体重が特定の値を上回っている場合に、対応するデータを別のデータセットに書き出すことができます。

ルックアップキーの数とデータセットのサイズによっては、ハッシュオブジェクトによる検索は、標準形式の検索よりも大幅に高速となります。

ハッシュオブジェクトの宣言とインスタンス作成

ハッシュオブジェクトを宣言するには DECLARE ステートメントを使用します。新しいハッシュオブジェクトを宣言した後、_NEW_演算子を使用して、同オブジェクトのインスタンスを作成します。次に例を示します。

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

この DECLARE ステートメントは、オブジェクト参照 MyHash がハッシュ型であることをコンパイラに伝えます。この時点では、宣言済みのオブジェクト参照は MyHash のみです。このオブジェクト参照は、ハッシュ型のコンポーネントオブジェクトを保持することができます。ハッシュオブジェクトの宣言は一度のみです。_NEW_演算子によりハッシュオブジェクトのインスタンスを作成し、同インスタンスをオブジェクト参照 MyHash に割り当てます。

コンポーネントオブジェクトの宣言とインスタンス作成を行う場合、前述した DECLARE ステートメントと_NEW_演算子を使用する 2 段階の方法以外にも、別の方法を使用できます。DECLARE ステートメントを使用して、コンポーネントオブジェクトの宣言とインスタンス作成の両方を一度に行うことができます。

```
declare hash myhash();
```

前述のステートメントは、次のコードと同じ意味を持ちます。

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

詳細については、“DECLARE Statement, Hash and Hash Iterator Objects” (*SAS Component Objects: Reference*)および“_NEW_ Operator, Hash or Hash Iterator Object” (*SAS Component Objects: Reference*)を参照してください。

コンストラクタを使用したハッシュオブジェクトデータの初期化

ハッシュオブジェクトの作成時に、データの初期化を行いたい場合があります。コンストラクタを使うと、ハッシュオブジェクトのインスタンスを作成し、そのハッシュオブジェクトのデータを初期化できます。

ハッシュオブジェクトコンストラクタは、次のいずれかの形式を持ちます。

- `declare hash` *object_name*`(`*argument_tag-1*`:` *value-1*
  `<,` *...argument_tag-n*`:` *value-n*`>);`
- *object_name* `= _new_ hash(`*argument_tag-1*`:` *value-1*
  `<,` *...argument_tag-n*`:` *value-n*`>);`

詳細については、“DECLARE Statement, Hash and Hash Iterator Objects” (*SAS Component Objects: Reference*)および“_NEW_ Operator, Hash or Hash Iterator Object” (*SAS Component Objects: Reference*)を参照してください。

キーとデータの定義

ハッシュオブジェクトは、ルックアップキーを使用してデータの保存や取得を行います。キーおよびデータは DATA ステップ変数です。ドット表記のメソッド呼び出しでこれらの DATA ステップ変数を使用することにより、ハッシュオブジェクトを初期化できます。キーを定義するには、対応するキー変数名を DEFINEKEY メソッドに渡します。データを定義するには、対応するデータ変数名を DEFINEDATA メソッドに渡します。すべてのキー変数およびデータ変数を定義した後、DEFINEDONE メソッドを呼び出します。キーおよびデータは、任意の数の文字または数値 DATA ステップ変数から構成されます。

たとえば、次のコードは、文字変数であるキー変数とデータ変数を初期化します。

```
length d $20;
length k $20;

if _N_ = 1 then do;
   declare hash h();
   rc = h.defineKey('k');
   rc = h.defineData('d');
   rc = h.defineDone();
end;
```

複数のキー変数およびデータ変数を持つことができますが、MULTIDATA:“YES”引数タグ付きでハッシュオブジェクトを作成していなければ、キー全体は一意でなければなりません。詳細については、“非一意キーとデータのペア” (511 ページ)を参照してください。

1 つのキーを使用して複数のデータ項目を保存できます。たとえば、キー変数とデータ変数に、それぞれ文字値とともに補助的な数値を保存するよう前述のプログラム例を変更できます。次の例では、キー変数とデータ変数を、それぞれ文字値と数値を 1 つずつ含むものとして定義しています。

```
length d1 8;
length d2 $20;
length k1 $20;
length k2 8;

if _N_ = 1 then do;
   declare hash h();
   rc = h.defineKey('k1', 'k2');
   rc = h.defineData('d1', 'd2');
   rc = h.defineDone();
end;
```

詳細については、“DEFINEDATA Method” (*SAS Component Objects: Reference*)、“DEFINEDONE Method” (*SAS Component Objects: Reference*)、および“DEFINEKEY Method” (*SAS Component Objects: Reference*)を参照してください。

注: ハッシュオブジェクトはキー変数に値を割り当てません。たとえば、**h.find(key:'abc')** >のようなメソッドを呼び出したとしても、SAS コンパイラは、ハッシュオブジェクトおよびハッシュ反復子により実行されたデータ変数の割り当てを検出できません。このため、プログラム内でキー変数またはデータ変数に値を割り当てていない場合、変数が初期化されていないというメッセージが発行され

ます。このようなメッセージを受け取らないようにするには、次のいずれかの措置を実施します。

- NONOTES システムオプションを設定します。
- 各キー変数および各データ変数に対する初期割り当てステートメント(通常、欠損値を割り当てるもの)を提供します。
- すべてのキー変数およびデータ変数をパラメータとする CALL MISSING ルーチンを使用します。次に例を示します。

```
length d $20;
length k $20;

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

非一意キーとデータのペア

デフォルトで、1 つのハッシュオブジェクト内のすべてのキーは一意となります。これは、各キーにつき、複数のデータ変数からなる 1 つの集合が存在することを意味します。状況によっては、ハッシュオブジェクト内で重複したキーを使用したい場合があります。複数のデータ変数からなる複数の集合が 1 つのキーに関連付けられている場合がこれに相当します。

たとえば、キーが患者 ID で、データが来院日であるとします。ある患者が何回も来院する場合、複数の来院日が 1 つの患者 ID に関連付けられることになります。MULTIDATA:“YES”という引数タグ付きでハッシュオブジェクトを作成すると、データ変数の複数の集合が 1 つのキーに関連付けられます。

データセットに重複したキーが含まれている場合、デフォルトでは、最初のインスタンスのみがハッシュオブジェクトに保存され、それ以降のインスタンスは無視されます。最終インスタンスをハッシュオブジェクトに格納するには、DUPLICATE 引数タグを使用します。DUPLICATE 引数タグを使用すると、重複キーが存在する場合、SAS ログにエラーが出力されます。

ただし、DECLARE ステートメントや_NEW_演算子で MULTIDATA 引数タグを使用しているならば、ハッシュオブジェクトはキーごとに複数の値を保存できます。ハッシュオブジェクトは、複数の値を、キーに関連付けられているリスト内に保持します。このリスト内のデータを調べるには、HAS_NEXT や FIND_NEXT などのメソッドを使用します。

複数のデータ項目を含むリストの内容を調べるには、現在のリスト項目を知る必要があります。まず、任意のキーに関して FIND メソッドを呼び出します。FIND メソッドにより、現在のリスト項目が設定されます。続いて、どのキーが複数のデータ値を持っているかを判定するために、HAS_NEXT メソッドを呼び出します。キーが別のデータ値を持っていると判定されたら、FIND_NEXT メソッドを使用してそのデータ値を取り出します。FIND_NEXT メソッドは、現在のリスト項目をリスト内の次の項目へと移動し、その項目に対応するデータ変数を設定します。

指定のキーを見つけるためにリスト内を前方向に移動するだけでなく、リスト内を後方向に移動を繰り返すこともできます。これを行うには、HAS_PREV および FIND_PREV の各メソッドを前述と同じように使用します。

1 つのキーに対して複数の値を持つハッシュオブジェクトがある場合、反復 DO ループで DO_OVER メソッドを使用して、重複キー全体をスキャンします。DO_OVER メソッドは、最初のメソッド呼び出しでキーを読み取り、重複キーリストの最後まで繰り返し処理を続けます。

注: 複数データ項目リストの項目は、挿入順序で管理されます。

非一意キーとデータのペアに関連するメソッドの詳細については、“Dictionary of Hash and Hash Iterator Object Language Elements” (*SAS Component Objects: Reference*)を参照してください。

データの保存と取得

データの保存方法と取得方法

ハッシュオブジェクトのキー変数とデータ変数を初期化した後、そのハッシュオブジェクトにデータを保存するには、ADD メソッドを使用します。または、*dataset* 引数タグを使用すると、データセットをそのハッシュオブジェクトにロードできます。*dataset* 引数タグを使用する場合、データセットに同じキーの値を持つ複数のオブザベーションが含まれているならば、デフォルトでは、ハッシュテーブル内の最初のオブザベーションが保持され、それ以降のオブザベーションは無視されます。最終インスタンスをハッシュオブジェクトに保存するか、または重複キーが存在する場合にエラーをログに書き出すには、DUPLICATE 引数タグを使用します。各キーで重複した値を許可するには、MULTIDATA 引数タグを使用します。

その後、各キーに 1 つのデータ値が存在する場合には、FIND メソッドを使用してハッシュオブジェクトを検索し、そのハッシュオブジェクトからデータを取り出すことができます。各キーに複数のデータ項目が存在する場合、データの検索と取り出しを行うには FIND_NEXT メソッドおよび FIND_PREV メソッドを使用します。

詳細については、“ADD Method” (*SAS Component Objects: Reference*)、“FIND Method” (*SAS Component Objects: Reference*)、“FIND_NEXT Method” (*SAS Component Objects: Reference*)、および“FIND_PREV Method” (*SAS Component Objects: Reference*)を参照してください。

REF メソッドを使うと、FIND メソッドと ADD メソッドの操作を一度に行うことができます。たとえば、次のコード量を減らせます。

```
rc = h.find();
  if (rc != 0) then
    rc = h.add();
```

そのためには次の単一のメソッド呼び出しに書き換えます。

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

詳細については、“REF Method” (*SAS Component Objects: Reference*)を参照してください。

注: また、ハッシュ反復子オブジェクトを使用して、一度に 1 データ項目ずつ、ハッシュオブジェクトデータを順方向および逆方向に取り出すこともできます。詳細については、“ハッシュ反復子オブジェクトの使用 ” (521 ページ)を参照してください

例 1:ADD メソッドと FIND メソッドを使用したデータの保存と取得

次の例では、ADD メソッドを使用して、ハッシュオブジェクトにデータを保存し、データをキーに関連付けています。続いて、FIND メソッドを使って、キー値 **Homer** に関連付けられているデータを取得します。

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

```
length k $20;

/* Declare the hash object and key and data variables */
if _N_ = 1 then do;
   declare hash h();
   rc = h.defineKey('k');
   rc = h.defineData('d');
   rc = h.defineDone();
end;

/* Define constant value for key and data */
k = 'Homer';
d = 'Odyssey';
/* Use the ADD method to add the key and data to the hash object */
rc =h.add();
if (rc ne 0) then
   put 'Add failed.';

/* Define constant value for key and data */
k = 'Joyce';
d = 'Ulysses';
/* Use the ADD method to add the key and data to the hash object */
rc = h.add();
if (rc ne 0) then
   put 'Add failed.';

k = 'Homer';
/* Use the FIND method to retrieve the data associated with 'Homer' key */
rc = h.find();
if (rc = 0) then
   put d=;
else
   put 'Key Homer not found.';
run;
```

この FIND メソッドは、データ値 **Odyssey** (キー値 **Homer** に関連付けられている)を変数 D に割り当てます。

例 2: データセットのロードと、FIND メソッドを使用したデータの取得

データセット Small には数値変数 K (キー)と S (データ)が、もう 1 つのデータセット LARGE には対応するキー変数 K がそれぞれ含まれているものとします。次のコードは、Small データセットをハッシュオブジェクトにロードした後、そのハッシュオブジェクトを検索することにより、LARGE データセット内の変数 K の値にマッチするキーを取り出します。

```
data match;
   length k 8;
   length s 8;
   if _N_ = 1 then do;
      /* load SMALL data set into the hash object */
     declare hash h(dataset: "work.small";
      /* define SMALL data set variable K as key and S as value */
      h.defineKey('k');
      h.defineData('s');
      h.defineDone();
      /* avoid uninitialized variable notes */
```

```
      call missing(k, s);
   end;

/* use the SET statement to iterate over the LARGE data set using */
/* keys in the LARGE data set to match keys in the hash object */
set large;
rc = h.find();
if (rc = 0) then output;
run;
```

dataset 引数タグには、Small データセットが指定されています。このデータセットのキーとデータが、DEFINEDONE メソッドの実行時にハッシュオブジェクトにより読み込まれロードされます。.続いて、FIND メソッドを使用してデータを取得します。

キーサマリーの管理

ハッシュオブジェクトキーのサマリーカウントを管理するには、ハッシュオブジェクトの宣言時に SUMINC 引数タグを使用します。このタグ値は文字列式であり、数値 DATA ステップ変数の名前である SUMINC へと展開されます。

この SUMINC タグは、ハッシュオブジェクトに各キーのサマリー値を管理するための内部ストレージを割り当てるよう命じます。

ハッシュキーのサマリー値は、ADD メソッドや REPLACE メソッドが使用された場合にはいつでも、SUMINC 変数の値へ初期化されます。

ハッシュキーのサマリー値は、FIND、CHECK、REF の各メソッドのいずれかが使用された場合にはいつでも、SUMINC 変数の値だけ増分されます。

SUMINC 値は、負数値、正数値、ゼロのいずれかになります。同変数値は整数でなくてもかまいません。キーの SUMINC 値はデフォルトではゼロになります。

次の例では、最初の ADD メソッドを呼び出す前に、K=99 のサマリーカウントを 1 に設定しています。その後、新しい COUNT 値が与えられるたびに、続く FIND メソッドにより、その値がキーサマリーに加算されます。この例では、各キーにつき 1 つのデータ値が存在します。SUM メソッドにより、キーサマリーの現在の値が取り出され、その値が DATA ステップ変数 TOTAL に格納されます。各キーにつき複数の項目が存在する場合、SUMDUP メソッドによりキーサマリーの現在の値が取得されます。

```
data _null_;
 length k count 8;
 length total 8;
 dcl hash myhash(suminc: 'count');
 myhash.defineKey('k');
 myhash.defineDone();

 k = 99;
 count = 1;
 myhash.add();


/* COUNT is given the value 2.5 and the */
/* FIND sets the summary to 3.5*/
 count = 2.5;
 myhash.find();

/* The COUNT of 3 is added to the FIND and */
/* sets the summary to 6.5. */
 count = 3;
```

```
 myhash.find();


/* The COUNT of -1 sets the summary to 5.5. */
 count = -1;
 myhash.find();

/* The SUM method gives the current value of */
/* the key summary to the variable TOTAL. */
 myhash.sum(sum: total);

/* The PUT statement prints total=5.5 in the log. */
 put total=;
 run;
```

次の例は、キー値が K=99 および K=100 である場合のサマリーをそれぞれ出力します。

```
 k = 99;
 count = 1;
 myhash.add();
 /* key=99 summary is now 1 */

 k = 100;
 myhash.add();
 /* key=100 summary is now 1 */

 k = 99;
 myhash.find();
 /* key=99 summary is now 2 */

 count = 2;
 myhash.find();
 /* key=99 summary is now 4 */

 k = 100;
 myhash.find();
 /* key=100 summary is now 3 */

 myhash.sum(sum: total);
 put 'total for key 100 = 'total;

 k = 99;

myhash.sum(sum:total);
 put 'total for key 99 = ' total;
```

最初の PUT ステートメントは、K=100 である場合のサマリーを出力します。

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

2 番目の PUT ステートメントは、K=99 である場合のサマリーを出力します。

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

キーサマリーは *dataset* 引数タグと組み合わせて使用できます。DEFINEDONE メソッドを使用してデータセットをハッシュオブジェクトに読み込む際に、すべてのキーサマリーが SUMINC 値に設定されます。続いて、FIND、CHECK、ADD の各メソッドを呼び出すと、対応するキーサマリーが変更されます。

```
declare hash myhash(suminc: "keycount", dataset: "work.mydata");
```

キーサマリーを使用して、指定のキーのオカレンス数をカウントできます。次の例では、データセット MyData をハッシュオブジェクトにロードした後、キーサマリーを使用して各キーのオカレンス数をデータセット Keys に保存しています。(SUMINC 変数が特定の値に設定されていないため、デフォルトの初期値であるゼロが使用されます。)

```
data mydata;
 input key;
datalines;
1
2
3
4
5
;
run;


data keys;
 input key;
datalines;
1
2
1
3
5
2
3
2
4
1
5
1
;
run;

data count;
 length total key 8;
 keep key total;

 declare hash myhash(suminc: "count", dataset:"mydata");
 myhash.defineKey('key');
 myhash.defineDone();
 count = 1;

 do while (not done);
   set keys end=done;
   rc = myhash.find();
 end;

 done = 0;
 do while (not done);
   set mydata end=done;
   rc = myhash.sum(sum: total);
   output;
 end;
```

```
    stop;
   run;
```

結果として作成されるデータセットの出力は次のようになります。

アウトプット 22.1 *キーサマリーの出力*

The SAS System

Obs	total	key
1	4	1
2	3	2
3	2	3
4	1	4
5	2	5

注: DECLARE ステートメントまたは_NEW_演算子の KEYSUM コンストラクタで、すべてのキーのキーサマリーを追跡する変数が宣言されます。KEYSUM 変数は出力データセットの一部であり、キーに対して 1 つ以上のデータ項目が存在する場合に機能します。

詳細については、“SUM Method” (*SAS Component Objects: Reference*)および“SUMDUP Method” (*SAS Component Objects: Reference*)を参照してください。

ハッシュオブジェクト内のデータの置換と削除

ハッシュオブジェクト内に保存されているデータの置換や削除を行うには、次のいずれかのメソッドを使用します。

- すべてのデータ項目を削除する場合、REMOVE メソッドを使用します。
- すべてのデータ項目を置換する場合、REPLACE メソッドを使用します。
- 現在のデータ項目のみを削除する場合、REMOVEDUP メソッドを使用します。
- 現在のデータ項目のみを置換する場合、REPLACEDUP メソッドを使用します。

次の例では、REPLACE メソッドを使用してデータ **`Odyssey`** を **`Iliad`** に置換した後、REMOVE メソッドを使用して、キー値 **`Joyce`** に関連付けられているすべてのデータをハッシュオブジェクトから削除しています。

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

/* Declare the hash object and key and data variables */
if _N_ = 1 then do;
   declare hash h();
   rc = h.defineKey('k');
   rc = h.defineData('d');
   rc = h.defineDone();
end;

/* Define constant value for key and data */
```

```
k = 'Joyce';
d = 'Ulysses';
/* Use the ADD method to add the key and data to the hash object */
rc = h.add();
if (rc ne 0) then
   put 'Add failed.';

/* Define constant value for key and data */
k = 'Homer';
d = 'Odyssey';
/* Use the ADD method to add the key and data to the hash object */
rc = h.add();
if (rc ne 0) then
   put 'Add failed.';

/* Use the REPLACE method to replace 'Odyssey' with 'Iliad' */
k = 'Homer';
d = 'Iliad';
rc = h.replace();
if (rc = 0) then
   put d=;
else
   put 'Replace not successful.';

/* Use the REMOVE method to remove the 'Joyce' key and data */
k = 'Joyce';
rc = h.remove();
if (rc = 0) then
   put k 'removed from hash object';
else
   put 'Deletion not successful.';

run;
```

次の行が SAS ログに書き出されます。

```
d=Iliad
Joyce removed from hash object
```

注: 関連付けられているハッシュ反復子が特定のキーを指している場合、REMOVE はそのキーまたはデータをハッシュオブジェクトから削除しません。その場合、エラーメッセージがログに書き出されます。

詳細については、“REMOVE Method” (*SAS Component Objects: Reference*)、“REMOVEDUP Method” (*SAS Component Objects: Reference*)、“REPLACE Method” (*SAS Component Objects: Reference*)、および“REPLACEDUP Method” (*SAS Component Objects: Reference*)を参照してください。

データセットへのハッシュオブジェクトデータの保存

指定のハッシュオブジェクト内にあるデータを含むデータセットを作成するには、OUTPUT メソッドを使用します。次の例では、2 つのキーとデータをハッシュオブジェクトに追加した後、それらのデータを Work.Out データセットに出力しています。

```
options pageno=1 nodate;

data test;
length d1 8;
```

```
length d2 $20;
length k1 $20;
length k2 8;

/* Declare the hash object and two key and data variables */
if _N_ = 1 then do;
   declare hash h();
   rc = h.defineKey('k1', 'k2');
   rc = h.defineData('d1', 'd2');
   rc = h.defineDone();
end;

/* Define constant value for key and data */
k1 = 'Joyce';
k2 = 1001;
d1 = 3;
d2 = 'Ulysses';
rc = h.add();

/* Define constant value for key and data */
k1 = 'Homer';
k2 = 1002;
d1 = 5;
d2 = 'Odyssey';
rc = h.add();

/* Use the OUTPUT method to save the hash object data to the OUT data set */
rc = h.output(dataset: "work.out");
run;

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

PROC PRINT により生成されるレポート出力を次に示します。

アウトプット 22.2 *ハッシュオブジェクトからデータセットを作成*

The SAS System

Obs	d1	d2
1	5	Odyssey
2	3	Ulysses

ハッシュオブジェクトキーは、出力データセットには保存されないことに注意してください。キーを出力データセットに含めたい場合は、DEFINEDATA メソッドを使ってそれらのキーをデータとして定義する必要があります。これを行うには、前述の DEFINEDATA メソッドを次のように書き換えます。

```
rc = h.defineData('k1', 'k2', 'd1', 'd2');
```

詳細については、“OUTPUT Method” (*SAS Component Objects: Reference*)を参照してください。

ハッシュオブジェクトの比較

あるハッシュオブジェクトを別のハッシュオブジェクトと比較するには、EQUALS メソッドを使用します。次の例では、2 つのハッシュオブジェクトを比較しています。EQUALS メソッドに 2 つの引数タグが指定されていることに注意してください。HASH 引数タグは 2 番目のハッシュオブジェクト名を表します。RESULTS 引数タグは比較結果(等しい場合は 1、等しくない場合は 0)を保持する数値変数名を表します。

```
length eq k 8;

declare hash myhash1();
myhash1.defineKey('k');
myhash1.defineDone();

declare hash myhash2();
myhash2.defineKey('k');
myhash2.defineDone();

rc = myhash1.equals(hash: 'myhash2', result: eq);
```

詳細については、“EQUALS Method” (*SAS Component Objects: Reference*)を参照してください。

ハッシュオブジェクト属性の使用

DATA ステップコンポーネントインターフェイスを使うと、属性を使用してハッシュオブジェクトから情報を取得できます。属性を指定するには次の構文を使用します。

```
attribute_value=obj.attribute_name;
```

ハッシュオブジェクトで使用できる属性は 2 つあります。NUM_ITEMS はハッシュオブジェクト内にある項目の数を、ITEM_SIZE は各項目のサイズを(バイト数で)それぞれ返します。次の例では、ハッシュオブジェクト内にある項目の数を取得しています。

```
n = myhash.num_items;
```

次の例では、ハッシュオブジェクト内の項目のサイズを取得しています。

```
s = myhash.item_size;
```

ITEM_SIZE 属性と NUM_ITEMS 属性を使用すると、ハッシュオブジェクトが使用しているメモリ量を概算できます。ITEM_SIZE 属性はハッシュオブジェクトが必要とする初期オーバーヘッドを反映しておらず、また必要となる内部アラインメントも考慮していません。このため、ITEM_SIZE を使用することで提供できるのはあくまでも近似値であり、正確なメモリ使用量は算出できません。

詳細については、“NUM_ITEMS Attribute ” (*SAS Component Objects: Reference*)および“ITEM_SIZE Attribute” (*SAS Component Objects: Reference*)を参照してください。

ハッシュ反復子オブジェクトの使用

ハッシュ反復子オブジェクトについて

ハッシュ反復子オブジェクトを使用すると、ルックアップキーに基づいてデータの保存や検索が行えます。ハッシュ反復子オブジェクトを使うと、ハッシュオブジェクトデータを順方向または逆方向に検索してキーを見つけることができます。

ハッシュ反復子オブジェクトの宣言とインスタンス作成

ハッシュ反復子オブジェクトを宣言するには、DECLARE ステートメントを使用します。新しいハッシュ反復子オブジェクトを宣言した後、_NEW_演算子を使用して、そのオブジェクトのインスタンスを作成します。ハッシュオブジェクト名を引数タグとして使用します。たとえば、次のようになります。

```
declare hiter myiter;
myiter = _new_ hiter('h');
```

DECLARE ステートメントは、オブジェクト参照 MyIter がハッシュ反復子型であることをコンパイラに指示します。この時点では、宣言済みのオブジェクト参照は MyIter のみです。このオブジェクト参照は、ハッシュ反復子型のコンポーネントオブジェクトを保持することができます。ハッシュ反復子オブジェクトの宣言は一度のみです。_NEW_演算子によりハッシュ反復子オブジェクトのインスタンスを作成し、同インスタンスをオブジェクト参照 MyIter に割り当てます。ハッシュオブジェクト H は、コンストラクタ引数として渡されます。このハッシュオブジェクト(ハッシュオブジェクト変数ではない)が、ハッシュ反復子に対して特別に割り当てられます。

ハッシュ反復子オブジェクトの宣言とインスタンス作成を行う場合、前述した DECLARE ステートメントと_NEW_演算子を使用する 2 段階の方法以外に、DECLARE ステートメントをコンストラクタメソッドとして使用することによりハッシュ反復子オブジェクトの宣言とインスタンス作成を一度に行うことができます。構文は次のようになります。

```
declare hiter object_name(hash_object_name);
```

前述の例では、ハッシュオブジェクト名を一重引用符または二重引用符で囲む必要があります。

たとえば、次のようになります。

```
declare hiter myiter('h');
```

前述のステートメントは、次のコードと同じ意味を持ちます。

```
declare hiter myiter;
myiter = _new_ hiter('h');
```

注: ハッシュ反復子オブジェクトを作成する前に、ハッシュオブジェクトの宣言とインスタンス作成を行う必要があります。詳細については、“ハッシュオブジェクトの宣言とインスタンス作成” (509 ページ)を参照してください。

たとえば、次のようになります。

```
if _N_ = 1 then do;
   length key $10;
   declare hash myhash(dataset:"work.x", ordered: 'yes');
   declare hiter myiter('myhash');
```

```
   myhash.defineKey('key');
   myhash.defineDone();
end;
```

このプログラムは、ハッシュ反復子オブジェクトのインスタンスを MyIter という変数名で作成します。ハッシュオブジェクト MyHash は、コンストラクタ引数として渡されます。ハッシュオブジェクトは ORDERED 引数タグを**'yes'**に設定して作成されているため、昇順のキー値でデータが返されます。

DECLARE ステートメントと_NEW_演算子の詳細については、*SAS Statements: Reference* を参照してください。

例:ハッシュ反復子を使用したハッシュオブジェクトデータの取得

次のコードでは、天文データを含むデータセット ASTRO を使用して、赤経(RA)の値が 12 を超えるメシエ天体(OBJ)を含むデータセットを作成しています。FIRST メソッドおよび NEXT メソッドを使用して、データを昇順で検索しています。FIRST メソッドおよび NEXT メソッドの詳細については、*SAS コンポーネントオブジェクト: リファレンス*を参照してください。

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

data out;
   if _N_ = 1 then do;
      length obj $10;
      length ra $10;
      length dec $10;
      /* Read ASTRO data set and store in asc order in hash obj */
```

```
      declare hash h(dataset:"work.astro", ordered: 'yes');
      /* Define variables RA and OBJ as key and data for hash object */
      declare hiter iter('h');
      h.defineKey('ra');
      h.defineData('ra', 'obj');
      h.defineDone();
      /* Avoid uninitialized variable notes */
      call missing(obj, ra, dec);
   end;
/* Retrieve RA values in ascending order */
rc = iter.first();
do while (rc = 0);
/* Find hash object keys greater than 12 and output data */
   if ra GE '12' then
      output;
   rc = iter.next();
end;
run;

proc print data=work.out;
   var ra obj;
   title 'Messier Objects Greater than 12 Sorted by Right Ascension Values';
run;
```

PROC PRINT により生成されるレポート出力を次に示します。

Messier Objects Greater than 12 Sorted by Right Ascension Values

Obs	ra	obj
1	12 13.8	M98
2	12 22.9	M100
3	12 29.8	M49
4	12 39.5	M68
5	12 42.0	M59
6	13 29.9	M51
7	13 42.2	M3
8	16 41.7	M13
9	16 57.1	M10
10	17 37.6	M14
11	17 56.8	M23
12	18 20.8	M17
13	18 31.6	M25
14	18 36.4	M22
15	18 53.6	M57
16	19 53.8	M71
17	20 23.9	M29
18	21 32.2	M39

Java オブジェクトの使用

Java オブジェクトについて

Java オブジェクトは、Java Native Interface (JNI)と同様のメカニズムを提供することにより、Java クラスのインスタンス作成や、結果として得られるオブジェクトのフィールドやメソッドへのアクセスを可能にします。Java プログラムと DATA ステッププログラムの両方を含むハイブリッドアプリケーションを作成できます。

CLASSPATH オプションと Java オプション

SAS の以前のバージョンでは、Java クラスを検出するためには、JREOPTIONS システムオプションを使用する必要がありました。

SAS では、環境変数 CLASSPATH を設定して、Java オブジェクトがユーザーの使用している Java クラスを検出できるようにする必要があります。Java オブジェクトは、現在の Java クラスパス内で検出された Java クラスのインスタンスを表します。ユーザーが使用するすべてのクラスは同クラスパスに存在する必要があります。クラスが.jar ファイル内に存在する場合、その.jar ファイル名がクラスパスに存在する必要があります。

環境変数 CLASSPATH の設定方法は、使用する動作環境によって異なります。多くのオペレーティングシステムでは、CLASSPATH 環境変数を、ローカル(ユーザーの SAS セッション内でのみの使用)、またはグローバルに設定できます。表 22.1 (525 ページ)に、各動作環境のメソッドと例を示します。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

表 22.1 *各動作環境での CLASSPATH 環境変数の設定*

動作環境	方法	例
Windows		
グローバル	コントロールパネルにおける Windows システムの環境変数	**コントロールパネル** ⇨ **システム** ⇨ **詳細設定** ⇨ **環境変数**(Windows XP のクラシックビューの場合)
	SAS 設定ファイル	`set classpath c:\HelloWorld.jar`
ローカル	SAS コマンドライン	`-set classpath c:\HelloWorld.jar`
UNIX		
グローバル	SAS 設定ファイル	`set classpath ~/HelloWorld.jar`
ローカル	EXPORT コマンド*	`export classpath=~/HelloWorld.jar;`
z/OS		
グローバル	TKMSENV データセット	`set TKJNI_OPT_CLASSPATH=/u/userid/java: /u/userid/java/test.jar: asis`
ローカル	利用不可	
VMS		
グローバル	コマンドライン**	`$ define java$classpath disk:[subdir] abc.jar, disk:[subdir2]def.jar`
	detach_template.com スクリプト(インストール時に **sas$root:[misc.base]** 内に作成されるもの)	`define java$classpath disk:[subdir] abc.jar, disk:[subdir2]def.jar`
ローカル	利用不可	

* 構文はシェルによって異なります。

** コマンドラインは、SAS System を呼び出す前に定義する必要があります。これにより、JVM が実際に稼働しているプロセスが、その定義を取得できるようになります。

Java オブジェクトの使用の制限事項と必要条件

Java オブジェクトを使用する場合、次の制限事項と必要条件が適用されます。

- Java オブジェクトが、SAS System から Java メソッドを呼び出すように設計されていること。Java オブジェクトが、SAS ライブラリの関数の拡張を意図していないこと。特に大きなデータセットを取り扱う場合、DATA ステップを高速にするには、PROC FCMP 関数を呼び出す方が効率的です。Java オブジェクトを使用して大きなデータセットを取り扱うタイプの処理を行うと、長い時間がかかります。
- SAS System がサポートしている Java ランタイム環境(JRE)は、SAS ソフトウェアのインストール時に明示的に必要とされる JRE のみになります。
- SAS System がサポートしている Java オプションは、SAS ソフトウェアのインストール時に設定される同オプションのみになります。
- 自分の Java アプリケーションを Java オブジェクトで使用する前に、同アプリケーションが正しく動作することを確認してください。
- SAS ログへの Java によるテキスト出力の先頭バイトにパーセント文字(%)を配置することは、SAS System により予約されています。Java のテキスト行の先頭バイトに%を出力する必要がある場合、もう 1 つのパーセント文字を付加することにより%をエスケープする必要があります(すなわち%%のように記述します)。

Java オブジェクトの宣言とインスタンス作成

Java オブジェクトを宣言するには DECLARE ステートメントを使用します。新しい Java オブジェクトを宣言した後、同オブジェクトをインスタンス化するには、その Java オブジェクト名を引数タグとして含む _NEW_ 演算子を使用します。

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

この例では、DECLARE ステートメントは、オブジェクト参照 J が Java 型であることをコンパイラに指示しています。これは、Java オブジェクトのインスタンスが変数 J に保存されることを意味します。この時点では、宣言済みのオブジェクト参照は J のみです。このオブジェクト参照は、Java 型のコンポーネントオブジェクトを保持することができます。Java オブジェクトの宣言は一度のみです。_NEW_ 演算子により Java オブジェクトのインスタンスを作成し、同インスタンスをオブジェクト参照 J に割り当てます。Java クラス名 SOMEJAVACLASS は、コンストラクタ引数として渡されます。これは Java オブジェクトコンストラクタで必要となる唯一の第 1 引数です。その他のすべての引数は、Java クラス自体のコンストラクタ引数になります。

Java オブジェクトの宣言とインスタンス化を行う場合、前述した DECLARE ステートメントと_NEW_演算子を使用する 2 段階の方法以外に、DECLARE ステートメントをコンストラクタメソッドとして使用することにより Java オブジェクトの宣言とインスタンス化を一度に行うことができます。構文は次のようになります。

DECLARE JAVAOBJ*object-name*(“*java-class*”, <*argument-1*, … *argument–n*>);

詳細については、“DECLARE Statement, Java Object” (*SAS Component Objects: Reference*)および“_NEW_ Operator, Java Object” (*SAS Component Objects: Reference*)を参照してください。

オブジェクトフィールドへのアクセス

Java オブジェクトのインスタンスを作成した後、その Java オブジェクトにおけるメソッド呼び出しを通じて、DATA ステップ内で同オブジェクトのパブリックフィールドおよびクラ

スフィールドにアクセスすることや、それらのフィールドを変更することができます。パブリックフィールドとは、Java クラスで public として宣言されている非静的なフィールドです。クラスフィールドとは、Java クラスからアクセスできる静的なフィールドです。

オブジェクトフィールドにアクセスするためのメソッド呼び出しは、非静的または静的フィールドのどちらにアクセスするかに応じて、次のいずれかの形式を使用します。

GET*type***FIELD**(“*field-name*”, *value*);

GETSTATIC*type***FIELD**(“*field-name*”, *value*);

オブジェクトフィールドを変更するためのメソッド呼び出しは、非静的または静的フィールドのどちらにアクセスするかに応じて、次のいずれかの形式を使用します。

SET*type***FIELD**(“*field-name*”, *value*);

SETSTATIC*type***FIELD**(“*field-name*”, *value*);

注: *type* 引数は Java のデータ型を表します。Java のデータ型と SAS のデータ型の関連性の詳細については、“データ型に関する問題” (527 ページ)を参照してください。*field-name* 引数は Java フィールドのタイプを指定します。*value* は、このメソッドにより返される(または設定される)値を指定します。

詳細については、“Dictionary of Java Object Language Elements” (*SAS Component Objects: Reference*)を参照してください。

オブジェクトメソッドへのアクセス

Java オブジェクトのインスタンスを作成した後、その Java オブジェクトにおけるメソッド呼び出しを通じて、DATA ステップ内で同オブジェクトのパブリックメソッドおよびクラスメソッドにアクセスすることができます。パブリックメソッドとは、Java クラスで public として宣言されている非静的なメソッドです。クラスメソッドとは、Java クラスからアクセスできる静的なフィールドです。

Java メソッドにアクセスするためのメソッド呼び出しは、非静的または静的メソッドのどちらにアクセスするかに応じて、次のいずれかの形式を使用します。

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

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

注: *type* 引数は Java のデータ型を表します。Java のデータ型と SAS のデータ型の関連性の詳細については、“データ型に関する問題” (527 ページ)を参照してください。

詳細については、“Dictionary of Java Object Language Elements” (*SAS Component Objects: Reference*)を参照してください。

データ型に関する問題

Java データ型セットとは、SAS データ型のスーパーセットです。Java には、標準的な数値および文字値に加えて、BYTE、SHORT、CHAR のようなデータ型があります。これに対して、SAS System には数値と文字という 2 つのデータ型しかありません。

Java オブジェクトのメソッドの呼び出し時に、Java データ型が SAS データ型へとどのようにマッピングされるかを次の表に示します。

表 22.2 Java データ型の SAS データ型へのマッピング

Java データ型	SAS データ型
BOOLEAN	数値
BYTE	数値
CHAR	数値
DOUBLE	数値
FLOAT	数値
INT	数値
LONG	数値
SHORT	数値
STRING	文字*

* Java の string データ型は、UTF-8 文字列として SAS の文字データ型へとマッピングされます。

STRING データ型以外は、Java クラスから DATA ステップへとオブジェクトを返すことはできません。ただし、Java メソッドにオブジェクトを渡すことはできます。詳細については、“Java オブジェクトの引数を渡す” (531 ページ)を参照してください。

オブジェクトを返す Java メソッドの一部には、オブジェクト値を変換するラッパークラスを作成することにより取り扱いが可能となるものもあります。次の例では、Java のハッシュテーブルはオブジェクト値を返します。ただし、型変換を行う単純な Java ラッパークラスを作成することにより、DATA ステップから同ハッシュテーブルを使い続けることができます。その後、DATA ステップから **dhash** および **shash** の両クラスにアクセスできます。

```
/* Java code */
import java.util.*;

public class dhash
{
   private Hashtable  table;

   public dhash()
   {
      table = new Hashtable  ();
   }

   public void put(double key, double value)
   {
      table.put(new Double(key), new Double(value));
   }

   public double get(double key)
   {
      Double ret = table.get(new Double(key));
      return ret.doubleValue();
```

```
   }
}

import java.util.*;

public class shash
{
   private Hashtable  table;

   public shash()
  {
      table = new Hashtable  ();
   }

   public void put(double key, String value)
  {
      table.put(new Double(key), value);
  }

   public String get(double key)
   {
      return table.get(new Double(key));
   }
}
/* DATA step code */
data _null_;
   dcl javaobj sh('shash');
   dcl javaobj dh('dhash');
   length d 8;
   length s $20;

   do i = 1 to 10;
      dh.callvoidmethod('vput', i, i * 2);
   end;

   do i = 1 to 10;
      sh.callvoidmethod('put', i, 'abc' || left(trim(i)));  end;

   do i = 1 to 10;
      dh.calldoublemethod('get', i, d);
      sh.callstringmethod('get', i, s);
      put d= s=;
   end;
run;
```

次の行が SAS ログに書き出されます。

```
d=2 s=abc1
d=4 s=abc2
d=6 s=abc3
d=8 s=abc4
d=10 s=abc5
d=12 s=abc6
d=14 s=abc7
d=16 s=abc8
d=18 s=abc9
```

```
d=20 s=abc10
```

Java オブジェクトと配列

DATA ステップ配列を Java オブジェクトに渡すことができます。

次の例では、配列 `d` および `s` を Java オブジェクト `j` に渡しています。

```
/* Java code
*/
import java.util.*;
import java.lang.*;
class jtest
{
   public void dbl(double args[])
  {
      for(int i = 0; i < args.length; i++)
         System.out.println(args[i]);
  }

  public void str(String args[])
  {
     for(int i = 0; i < args.length; i++)
        System.out.println(args[i]);
  }
}
```

```
/* DATA Step code */
data _null_;
   dcl javaobj j("jtest");
   array s{3} $20 ("abc", "def", "ghi");
   array d{10} (1:10);
   j.callVoidMethod("dbl", d);
   j.callVoidMethod("str", s);
run;
```

次の行が SAS ログに書き出されます。

```
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
abc
def
ghi
```

1 次元配列のパラメータのみがサポートされます。ただし、配列は行を順序どおりに埋めていく方式で渡されるという事実を利用することで、多次元配列の引数を渡すことも可能です。Java プログラムでは、次元のインデックス作成を手動で処理する必要があります。これは、1 次元配列パラメータを宣言した後、サブ配列へのインデックス付けを行う必要があることを意味します。

Java オブジェクトの引数を渡す

Java クラスから DATA ステップへオブジェクトを返すことはできませんが、オブジェクトや文字列を Java クラスメソッドに渡すことは可能です。

たとえば、**java/util/Vector** とその反復子に関して次のラッパークラスがあるとします。

```
/* Java code */
import java.util.*;

class mVector extends Vector
{
   public mVector()
  {
     super();
  }

  public mVector(double d)
  {
     super((int)d);
  }

  public void addElement(String s)
  {
     addElement((Object)s);
  }
}

import java.util.*;
public class mIterator
{
   protected mVector m_v;
   protected Iterator iter;

   public mIterator(mVector v)
   {
      m_v = v;
      iter = v.iterator();
   }

  public boolean hasNext()
  {
      return iter.hasNext();
  }

  public String next()
  {
     String ret = null;
     ret = (String)iter.next();
     return ret;
  }
}
```

これらのラッパークラスは、型変換を実行する場合に便利です(例:**mVector** コンストラクタは DOUBLE 引数を受け取ります)。**java/util/Vector** のコンストラクタは整数

値を受け取りますが、DATA ステップには整数型が存在しないため、同コンストラクタをオーバーロードすることが必要となります。

次の DATA ステッププログラムでは、これらのクラスを使用しています。同プログラムでは、ベクトルを作成し、同ベクトルに値を入力した後、同ベクトルを反復子のコンストラクタに渡しています。続いて、そのベクトル内のすべての値をリストしています。ベクトルに値を入力した後、反復子を作成する必要があることに注意してください。反復子は、その作成時にベクトルの変更カウントのコピーを保持します。このカウントは、同ベクトルの現在の変更カウントと同期した状態に保たれる必要があります。ベクトルに値が入力される前に反復子が作成された場合、同プログラムは例外をスローします。

```
/* DATA step code */
data _null_;
   length b 8;
   length val $200;
   dcl javaobj v("mVector");

   v.callVoidMethod("addElement", "abc");
   v.callVoidMethod("addElement", "def");
   v.callVoidMethod("addElement", "ghi");
   dcl javaobj iter("mIterator", v);

   iter.callBooleanMethod("hasNext", b);
   do while(b);
      iter.callStringMethod("next", val);
      put val=;
      iter.callBooleanMethod("hasNext", b);
   end;

   m.delete();
   v.delete();
   iter.delete();
run;
```

次の行が SAS ログに書き出されます。

```
val=abc
val=def
val=ghi
```

オブジェクトを渡すことに関する現時点の制限の 1 つとして、JNI メソッドのルックアップルーチンが、与えられたシグネチャに基づいて完全なクラス検索を実施しないことが挙げられます。これは、次に示すプログラムでは、**mIterator** が **Vector** を受け取るように変更できないことを意味します。

```
/* Java code */
public mIterator(Vector v)
{
    m_v = v;
    iter = v.iterator();
}
```

mVector は **Vector** のサブクラスですが、メソッドのルックアップルーチンはコンストラクタを見つけられません。現時点での唯一の解決策は、新しいメソッドを追加するか、またはラッパークラスを作成することにより、これらのデータ型を管理することです。

Java 例外

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

EXCEPTIONCHECK メソッドは、メソッド呼び出し時に例外が発生したかどうかを判定するために使用されます。例外をスローするメソッドを呼び出す場合、呼び出し後に例外をチェックすることを強く推奨します。例外がスローされた場合、適切な処置を実施した後、EXCEPTIONCLEAR を使って例外をクリアする必要があります。

例外デバッグロギングのオン/オフを切り替えるには、EXCEPTIONDESCRIBE メソッドを使用します。例外デバッグロギングをオンにすると、例外情報が JVM 標準出力に書き出されます。デフォルトで、JVM 標準出力は SAS ログへとリダイレクトされます。例外デバッグロギングはデフォルトではオフになります。

詳細については、“EXCEPTIONCHECK Method” (*SAS Component Objects: Reference*)、“EXCEPTIONCLEAR Method” (*SAS Component Objects: Reference*)、および“EXCEPTIONDESCRIBE Method” (*SAS Component Objects: Reference*)を参照してください。

Java 標準出力

次に示すような標準出力に送られる Java でのステートメントからの出力は、デフォルトで SAS ログへ送られます。

```
System.out.println("hello");
```

SAS ログへと送られる Java 出力は、DATA ステップの終了時にフラッシュされます。フラッシュが行われると、DATA ステップが実行中に生成されたすべての出力の後に、Java 出力が書き出されます。これらの出力を同期し、実行された順に出力が書き出されるようにするには、FLUSHJAVAOUTPUT メソッドを使用します。

Java オブジェクトの例

例 1:単純な Java メソッドの呼び出し

次の Java クラスは、3 つの数を合計する単純なメソッドを呼び出します。

```
/* Java code */
class MyClass
{
   double compute(double x, double y, double z)
      {
         return (x + y + z);
      }
}

/* DATA step code */
data _null_;
   dcl javaobj j("MyClass");

   rc = j.callDoubleMethod("compute", 1, 2, 3, r);

   put rc= r=;
run;
```

次の行が SAS ログに書き出されます。

```
rc=0 rc=6
```

例 2:ユーザーインターフェイスの作成

Java コンポーネントへのアクセスメカニズムに加えて、Java オブジェクトを使用して簡単な Java ユーザーインターフェイスを作成できます。

次の Java クラスは、複数のボタンを備えた簡単なユーザーインターフェイスを作成します。また、このユーザーインターフェイスは、ユーザーが入力したボタン選択の順番を表す値のキューを保持します。

```
/* Java code */
import java.awt.*;
import java.util.*;
import java.awt.event.*;

class colorsUI extends Frame
{
  private Button red;
  private Button blue;
  private Button green;
  private Button quit;
  private Vector list;
  private boolean d;
  private colorsButtonListener cbl;

  public colorsUI()
  {
    d = false;
    list = new Vector();
    cbl = new colorsButtonListener();

    setBackground(Color.lightGray);
    setSize(320,100);
    setTitle("New Frame");
    setVisible(true);
    setLayout(new FlowLayout(FlowLayout.CENTER, 10, 15));
    addWindowListener(new colorsUIListener());

    red = new Button("Red");
    red.setBackground(Color.red);
    red.addActionListener(cbl);

    blue = new Button("Blue");
    blue.setBackground(Color.blue);
    blue.addActionListener(cbl);

    green = new Button("Green");
    green.setBackground(Color.green);
    green.addActionListener(cbl);

    quit = new Button("Quit");
    quit.setBackground(Color.yellow);
    quit.addActionListener(cbl);

    this.add(red);
    this.add(blue);
```

```
  this.add(green);
  this.add(quit);

  show();
}

public synchronized void enqueue(Object o)
{
  synchronized(list)
    {
      list.addElement(o);
      notify();
    }
}

public synchronized Object dequeue()
{
  try
    {
      while(list.isEmpty())
        wait();

      if (d)
        return null;

      synchronized(list)
        {
          Object ret = list.elementAt(0);
          list.removeElementAt(0);
          return ret;
        }
    }
  catch(Exception e)
    {
      return null;
    }
}

public String getNext()
{
  return (String)dequeue();
}

public boolean done()
{
  return d;
}

class colorsButtonListener implements ActionListener
{
   public void actionPerformed(ActionEvent e)
   {
      Button b;
      String l;
      b = (Button)e.getSource();
      l = b.getLabel();
```

```
            if ( l.equals("Quit") )
            {
               d = true;
               hide();
               l = "";
            }
            enqueue(l);
         }
      }

      class colorsUIListener extends WindowAdapter
      {
         public void windowClosing(WindowEvent e)
         {
            Window w;
            w = e.getWindow();
            d = true;
            enqueue("");
            w.hide();
         }
      }

      public static void main(String s[])
      {
          colorsUI cui;
          cui = new colorsUI();
      }
   }
   /* DATA step code */
   data colors;
      length s $10;
      length done 8;
      drop done;

      if (_n_ = 1) then do;
       /* Declare and instantiate colors object (from colorsUI.class) */
         dcl javaobj j("colorsUI");
      end;

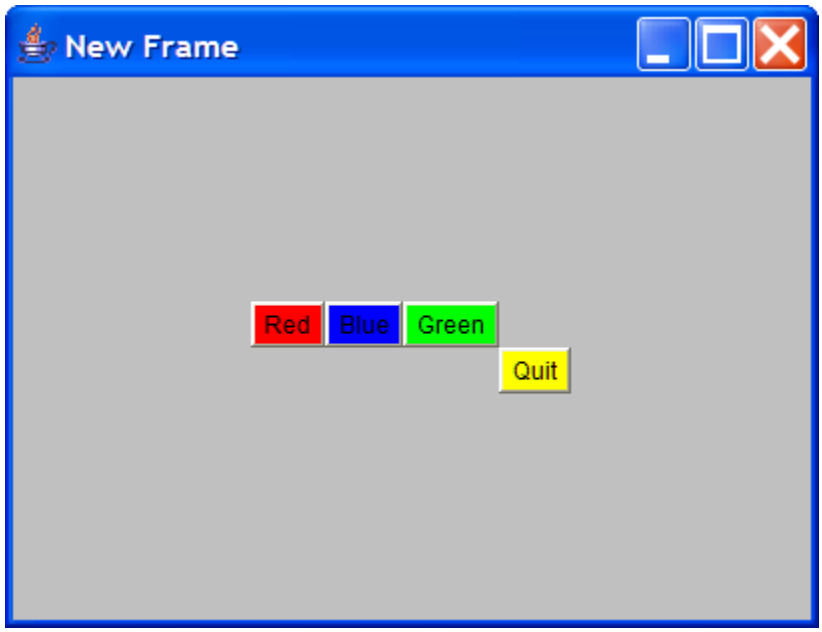
      /*
       * colorsUI.class will display a simple UI and maintain a
       * queue to hold color choices.
       */

      /* Loop until user hits quit button */
      do while (1);
         j.callBooleanMethod("done", done);
         if (done) then
            leave;
         else do;
          /* Get next color back from queue */
            j.callStringMethod("getNext", s);
            if s ne "" then
               output;
            end;
```

```
   end;
run;
proc print data=colors;
run;
quit;
```

DATA ステッププログラムで、**colorsUI** クラスのインスタンスを作成すると、対応するユーザーインターフェイスが表示されます。このユーザーインターフェイスのループを抜け出すには、**終了**をクリックします。この操作は、Done 変数を通じて DATA ステップに伝えられます。ループ中に、DATA ステップは当該 Java クラスのキューから値を取り出し、その値を次々に出力データセットへと書き出します。

図 22.1 *Java オブジェクトを使用して作成したユーザーインターフェイス*



例 3:カスタムクラスローダーの作成

すべての Java クラスをクラスパスに配置するとは限りません。ユーザーは、クラスを検出してロードする、ユーザー独自のクラスローダーを作成できます。カスタムクラスローダーの作成例を次に示します。

次の例では、クラス **x** を、フォルダ/ディレクトリ **y** 内に作成します。このクラス内のメソッドを呼び出すには、フォルダ **y** を含んでいるクラスパスを Java オブジェクトで指定します。

```
/* Java code */
package com.sas;

public class x
{
   public void m()
   {
      System.out.println("method m in y folder");
   }

   public void m2()
   {
      System.out.println("method m2 in y folder");
```

```
   }
}

/* DATA step code */
data _null_;
   dcl javaobj j('com/sas/x');
   j.callvoidmethod('m');
   j.callvoidmethod('m2');
run;
```

次の行が SAS ログに書き出されます。

```
method m in y folder
method m2 in y folder
```

次のプログラムでは、別のクラス **x** を、別のフォルダ **z** に保存しています。

```
/* Java code
*/
package com.sas;

public class z
{
   public void m()
   {
      System.out.println("method m in y folder");
   }

   public void m2()
   {
      System.out.println("method m2 in y folder");
   }
}
```

フォルダ **y** に保存されているクラスではなく、前述のクラス内のメソッドを呼び出すには、クラスパスを変更します。ただし、この変更を有効にするには SAS System を再起動する必要があります。次のメソッドを使うと、クラスをロードする方法をより動的に制御できます。

カスタムクラスローダーを作成するには、Java オブジェクトを通じて呼び出すメソッドのすべてを含むインターフェイスを作成します。次のプログラムでは、**m** および **m2** がそのようなメソッドになります。

```
/* Java
code */
public interface apiInterface
{
   public void m();
   public void m2();
}
```

続いて、それらのメソッドを実装したクラスを作成します。

```
/* Java code */
import com.sas.x;

public class apiImpl implements apiInterface
{
   private x x;

   public apiImpl()
```

```
   {
      x = new x();
   }

   public void m()
   {
      x.m();
   }

   public void m2()
   {
      x.m2();
   }
}
```

これらのメソッドは、Java オブジェクトインスタンスクラスへデリゲートすることにより呼び出されます。カスタムクラスローダー **apiClassLoader** を作成するプログラムについては、本セクションの末尾をご覧ください。

```
/* Java code */
public class api
{
  /* Load classes from the z folder */
  static ClassLoader customLoader =  new apiClassLoader("C:\\z");
  static String API_IMPL = "apiImpl";
  apiInterface cp = null;

  public api()
    {
      cp = load();
    }

  public void m()
    {
      cp.m();
    }

  public void m2()
    {
      cp.m2();
    }

  private static apiInterface load()
    {
      try
        {
          Class aClass = customLoader.loadClass(API_IMPL);
          return (apiInterface) aClass.newInstance();
        }
      catch (Exception e)
       {
          e.printStackTrace();
          return null;
        }
    }
}
```

次の DATA ステッププログラムでは、**api** という Java オブジェクトインスタンスクラスを通じてデリゲートすることにより、これらのメソッドを呼び出しています。この Java オブジェクトは、**api** クラスのインスタンスを作成します。これにより、**z** フォルダからクラスをロードするカスタムクラスローダーが作成されます。**api** クラスは、カスタムローダーを呼び出した後、**apiImpl** というインターフェイス実装クラスのインスタンスを同 Java オブジェクトに返します。メソッドが同 Java オブジェクトを通じて呼び出されると、**api** クラスは、それらのメソッドを実装クラスへデリゲートします。

```
/* DATA step code */
data _null_;
   dcl javaobj j('api');
   j.callvoidmethod('m');
   j.callvoidmethod('m2');
run;
```

次の行が SAS ログに書き出されます。

```
method m is z folder
method m2 in z folder
```

先の Java コードでは、.jar ファイルを使用することによっても、**ClassLoader** コンストラクタのクラスパスを増強できます。

```
static ClassLoader customLoader = new apiClassLoader("C:\\z;C:\\temp\some.jar");
```

この場合、カスタムクラスローダーの Java プログラムは次のようになります。ユーザーは、このプログラムを必要に応じて追加または変更できます。

```
import java.io.*;
import java.util.*;
import java.util.jar.*;
import java.util.zip.*;

public class apiClassLoader extends ClassLoader
{
  //class repository where findClass performs its search
  private List classRepository;

  public apiClassLoader(String loadPath)
    {
      super(apiClassLoader.class.getClassLoader());
      initLoader(loadPath);
    }

  public apiClassLoader(ClassLoader parent,String loadPath)
    {
      super(parent);
      initLoader(loadPath);
    }

  /**
   * This method will look for the class in the class repository. If
   * the method cannot find the class, the method will delegate to its parent
   * class loader.
   *
   * @param className A String specifying the class to be loaded
   * @return  A Class object loaded by the apiClassLoader
   * @throws ClassNotFoundException if the method is unable to load the class
   */
```

```
public Class loadClass(String name) throws ClassNotFoundException
  {
    // Check if the class is already loaded
    Class loadedClass = findLoadedClass(name);

    // Search for class in local repository before delegating
   if (loadedClass == null)
     {
       loadedClass = myFindClass(name);
     }

    // If class not found, delegate to parent
    if (loadedClass == null)
      {
        loadedClass = this.getClass().getClassLoader().loadClass(name);
      }
    return loadedClass;
  }

private Class myFindClass(String className) throws ClassNotFoundException
  {
    byte[] classBytes = loadFromCustomRepository(className);
    if(classBytes != null)
      {
        return defineClass(className,classBytes,0,classBytes.length);
      }
    return null;
  }

/**
 *  This method loads binary class file data from the classRepository.
 */
private byte[] loadFromCustomRepository(String classFileName)
  throws ClassNotFoundException
  {
    Iterator dirs = classRepository.iterator();
    byte[] classBytes = null;
    while (dirs.hasNext())
      {
        String dir = (String) dirs.next();

        if (dir.endsWith(".jar"))
          {
            // Look for class in jar

            String jclassFileName = classFileName;

            jclassFileName = jclassFileName.replace('.', '/');
            jclassFileName += ".class";

            try
              {
                JarFile j = new JarFile(dir);
                for (Enumeration e = j.entries(); e.hasMoreElements() ;)
                  {
                    Object n = e.nextElement();
```

```
                            if (jclassFileName.equals(n.toString()))
                              {
                                ZipEntry zipEntry = j.getEntry(jclassFileName);
                                if (zipEntry == null)
                                  {
                                    return null;
                                  }
                                else
                                  {
                                    // read file
                                    InputStream is = j.getInputStream(zipEntry);
                                    classBytes = new byte[is.available()];
                                    is.read(classBytes);
                                    break;
                                  }
                              }
                          }
                      }
                    catch (Exception e)
                      {
                        System.out.println("jar file exception");
                        return null;
                      }
                  }
                else
                  {
                    // Look for class in directory
                    String fclassFileName = classFileName;

                    fclassFileName = fclassFileName.replace('.', File.separatorChar);
                    fclassFileName += ".class";

                    try
                      {
                        File file = new File(dir,fclassFileName);
                        if(file.exists()) {
                          //read file
                          InputStream is = new FileInputStream(file);
                          classBytes = new byte[is.available()];
                          is.read(classBytes);
                          break;
                        }
                      }
                    catch(IOException ex)
                      {
                        System.out.println("IOException raised while reading class
file data");
                        ex.printStackTrace();
                        return null;
                      }
                  }
              }
            return classBytes;
        }
```

```
  private void initLoader(String loadPath)
    {
      /*
       * loadPath is passed in as a string of directories/jar files
       * separated by the File.pathSeparator
       */
      classRepository = new ArrayList();
      if((loadPath != null) && !(loadPath.equals("")))
        {
          StringTokenizer tokenizer =
            new StringTokenizer(loadPath,File.pathSeparator);
          while(tokenizer.hasMoreTokens())
            {
              classRepository.add(tokenizer.nextToken());
            }
        }
    }
}
```

23 章

配列処理

配列処理関係の用語

配列
　配列とは、DATA ステップ処理の中で、SAS 変数を一時的にグループ化する方法です。配列に指定した変数は特定の順序で配置され、配列名によって識別されます。配列は、現在の DATA ステップが処理されている間だけ存在します。配列名は、同一の DATA ステップ内にある他の配列と区別するための名称です。配列名は変数ではありません。

　注: SAS System の配列は、SAS System 以外のプログラミング言語における配列とは少し異なります。SAS System では、配列はデータ構造体ではなく、変数のグループを一時的に簡単に識別できるようにするための方法です。

配列処理
　配列として定義した一連の変数グループに対して、同じタスクを実行することです。

配列参照
　配列の要素を参照するための方法です。

1 次元配列
　定義した配列の形式が 1 次元の形式である、単純な変数グループです。

多次元配列
　定義した配列の形式が列と行などの 2 次元以上の形式である、より複雑な変数グループです。

配列処理の基本的なステップは、次のとおりです。

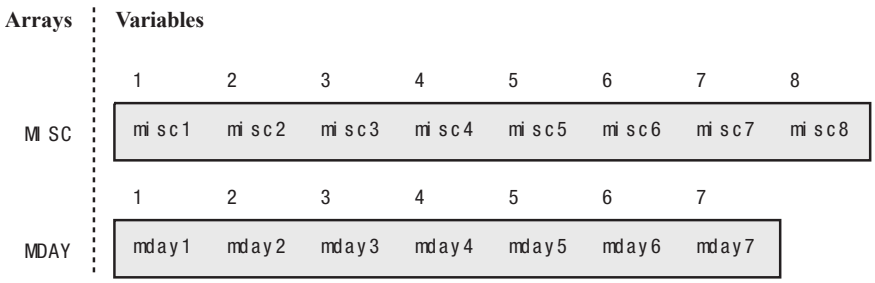
- 関連する一連の変数を配列に定義して、グループ化します。
- 配列処理を行う現在の変数を選択します。
- 処理を繰り返します。

配列処理の概念

1 次元配列

次の図は、Misc と Mday という 2 つの 1 次元配列を概念的に表したものです。

図 23.1 *1 次元配列*



配列 Misc には 8 つの要素が格納されています。これらは Misc1 から Misc8 までの変数です。変数に含まれるデータを参照するには、Misc{*n*}という形式を使用します。

ここで、*n* は配列内での要素番号です。たとえば、Misc{6}は配列内の 6 番目の要素になります。

配列 Mday には 7 つの要素が格納されています。これらは Mday1 から Mday7 までの変数です。Mday{3}は、配列内の 3 番目の要素になります。

2 次元配列

次の図は、Expenses という 2 次元配列を概念的に表したものです。

図 23.2 *2 次元配列の例*

First Dimension		**Second Dimension**							
Expense Categories		Days of the Week							Total
		1	2	3	4	5	6	7	8
Hotel	1	hotel1	hotel2	hotel3	hotel4	hotel5	hotel6	hotel7	hotel8
Phone	2	phone1	phone2	phone3	phone4	phone5	phone6	phone7	phone8
Pers. Auto	3	peraut1	peraut2	peraut3	peraut4	peraut5	peraut6	peraut7	peraut8
Rental Car	4	carrnt1	carrnt2	carrnt3	carrnt4	carrnt5	carrnt6	carrnt7	carrnt8
Airfare	5	airlin1	airlin2	airlin3	airlin4	airlin5	airlin6	airlin7	airlin8
Dues	6	dues1	dues2	dues3	dues4	dues5	dues6	dues7	dues8
Registration Fees	7	regfee1	regfee2	regfee3	regfee4	regfee5	regfee6	regfee7	regfee8
Other	8	other1	other2	other3	other4	other5	other6	other7	other8
Tips (non-meal)	9	tips1	tips2	tips3	tips4	tips5	tips6	tips7	tips8
Meals	10	meals1	meals2	meals3	meals4	meals5	meals6	meals7	meals8

配列 Expenses には 10 個のグループがあり、それぞれ 8 つの変数が格納されています。10 個のグループ(費用の科目)は、配列の 1 次元目を構成しています。8 つの変数(曜日)は、2 次元目を構成しています。配列変数内のデータを参照するには、Expenses{*m,n*}という形式を使用します。ここで、*m* は配列の 1 次元目の要素番号であり、*n* は配列の 2 次元目の要素番号です。Expenses{6,4}と記述すると、4 番目の曜日の税額(変数は Dues4)が参照されます。

配列の定義や参照を行う場合の構文

1 次元配列または多次元配列を定義するには、ARRAY ステートメントを使用します。ARRAY ステートメントの形式は次のとおりです。

ARRAY *array-name* {*number-of-elements*} <$> <*length*> <*array-elements*> <(*initial-value-list*)>;

ここで、

array-name
変数グループを識別する SAS 名を指定します。

number-of-elements
グループ内の変数の数を指定します。この値は、かっこ()、中かっこ{}、または大かっこ[]で囲む必要があります。

$
配列内の要素が文字要素であることを指定します。

length
長さが割り当てられていない配列内の要素の長さを指定します。

array-elements
グループ内の変数の名前リストです。配列に定義する変数は、すべて同じ種類(文字または数値)でなければなりません。

initial-value-list
配列内にある対応する要素の初期値のリストです。

詳細については、“ARRAY Statement” (*SAS Statements: Reference*)を参照してください。

同一の DATA ステップ内ですでに定義されている配列を参照するには、配列参照(Array Reference)ステートメントを使用します。配列参照ステートメントの形式は次のとおりです。

array-name {*subscript*}

ここで、

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
配列の要素を参照する添字を指定します。数値定数、数値変数、SAS 数式、アスタリスク(*)のいずれかを指定できます。

注: 配列の添字の下限は、SAS System 以外の一部のプログラミング言語では 0 ですが、SAS System のデフォルトでは 1 になります。

配列参照ステートメントの詳細については、*SAS Statements: Reference* にある同ステートメントの説明を参照してください。

1 次元配列の処理

1 次元配列でのグループ変数

次の ARRAY ステートメントは、Reference、Usage、Introduction という 3 つの変数を保持する配列 Books を定義しています。

```
array books{3} Reference Usage Introduction;
```

配列を定義すると、配列の各要素に *array reference* (配列参照)が *array-name*{*subscript*}という形式で割り当てられます。ここで、*subscript* (添字)はリスト内での変数の位置です。次の表は、前述の ARRAY ステートメントによって割り当てられる配列参照を示しています。

表 23.1 *配列 Books での配列参照の割り当て*

変数	配列参照
Reference	books{1}

変数	配列参照
Usage	books{2}
Introduction	books{3}

DATA ステップの後続の部分では、配列に含まれる変数を処理する場合、変数名または配列参照で各変数を参照できます。たとえば、変数名 Reference と配列参照 books{1}は同一の値を指します。

DO ループによる反復処理

同じ操作を繰り返して何回か実行するには、反復 DO ループを使用します。配列を処理するための単純な反復 DO ループは、次のような形式になります。

DO *index-variable*=1 **TO** *number-of-elements-in-array*;
... more SAS statements ...
END;

反復 DO ステートメントに記述された命令に従って、ループは反復処理されます。反復 DO ステートメントには *index-variable* (インデックス変数名)を指定します。インデックス変数の値は、ループが反復されるたびに変化します。

配列に含まれる変数の数だけループを実行するには、*index-variable* の値を 1 TO *number-of-elements-in-array* に指定します。*index-variable* の値は、ループの反復が新しく始まる前に毎回 1 ずつ増分します。値が *number-of-elements-in-array* を超えると、ループの処理が停止します。*index-variable* は、デフォルトでは自動的に出力データセットに格納されます。出力データセットにインデックス変数が書き込まれないようにするには、DROP ステートメントまたは DROP=データセットオプションを使用します。

count というインデックス変数を持ち、3 回実行される反復 DO ループの形式は次のようになります。

```
do count=1 to 3;
    … more SAS statements …
end;
```

ループが初めて処理されるときは、count の値は 1 です。2 回目は 2 になり、3 回目は 3 になります。4 回目の反復が開始される時点では、count の値は 4 になります。ここで、値が指定された範囲を超えるため、ループの処理は停止します。

DO ループによる特定の配列要素の処理

配列に含まれる特定の要素を処理するには、対象とする要素を反復 DO ステートメントの処理範囲として指定します。たとえば、次のステートメントを使用して、7 つの要素を含む Days という配列を作成します。

```
array days{7} D1-D7;
```

配列 Days にある特定の要素を選択して処理するには、次のような DO ステートメントを使用します。

表 23.2 DO ステートメントの処理

DO ステートメント	説明
`do i=2 to 4;`	要素 2 - 4 を処理します。
`do i=1 to 7 by 2;`	要素 1、3、5、7 を処理します。
`do i=3,5;`	要素 3 と 5 を処理します。

現在の変数の選択

配列内にある変数のうち、どの変数を反復 DO ループに使用するのかを指定する必要があります。配列に含まれる変数を識別するには、配列参照を使用します。配列参照の添字には、変数名、番号、式を使用できます。したがって、DO ループのインデックス変数が配列参照の添字になるようなプログラムステートメントを記述できます。たとえば、*array-name*{*index-variable*})と記述できます。インデックス変数の値が変化すると、配列参照の添字も変化します。同様に、参照されている変数も変化します。

次の例では、インデックス変数 count を DO ループ内で配列参照の添字として使用しています。

```
array books{3} Reference Usage Introduction;
do count=1 to 3;
   if books{count}=. then books{count}=0;
end;
```

count の値が 1 のときには、配列参照は Books{1}と解釈され、IF-THEN ステートメントは Books{1}で処理されます。Books{1}は変数 Reference です。count が 2 のときは、Books{2}でステートメントが処理されます。Books{2}は変数 Usage です。count が 3 のときは、Books{3}でステートメントが処理されます。Books{3}は変数 Introduction です。

この例にあるステートメントでは、次の処理を行います。

- DO ループに含まれる操作を 3 回実行します。
- IF-THEN ステートメントが反復を実行するたびに、配列の添字 count を count の現在の値で置き換えます。
- 配列参照を使用して変数を指定し、その変数に対して IF-THEN ステートメントを実行します。
- 条件が真の場合は、欠損値を 0 で置き換えます。

次の DATA ステップでは、配列 Book を定義し、それを DO ループで処理しています。

```
   options linesize=80 pagesize=60;

   data changed(drop=count);
      input Reference Usage Introduction;
      array book{3} Reference Usage Introduction;
      do count=1 to 3;
         if book{count}=. then book{count}=0;
      end;
      datalines;
   45 63 113
   .  75 150
```

```
62 .   98
;

proc print data=changed;
   title 'Number of Books Sold';
run;
```

次の出力結果は、データセット CHANGED を示しています。

アウトプット 23.1 *配列による欠損値の処理*

Number of Books Sold

Obs	Reference	Usage	Introduction
1	45	63	113
2	0	75	150
3	62	0	98

配列要素の数の定義

配列に含まれる要素の数を定義する場合は、アスタリスク(*)を大かっこ[ ]、中かっこ{ }、かっこ( )で囲んで[*]、{*}、(*)のようにして使用することで、要素の数をカウントしたり、要素の数を指定したりできます。アスタリスク(*)を使用して要素の数を指定する場合は、各配列要素を列挙する必要があります。次の例にある配列 C1Temp は、気温を保持する 5 つの変数を参照しています。

```
array c1temp{*} c1t1 c1t2 c1t3 c1t4 c1t5;
```

要素の数を明示的に指定する場合は、ARRAY ステートメントで変数または配列要素の名前を省略してもかまいません。省略した場合、変数名は、配列名と数字 1、2、3、...を連結して作成します。一連の変数名がすでに存在していた場合は、新しい変数は作成されず、既存の変数を使用します。次の例にある配列 c1t は、c1t1、c1t2、c1t3、c1t4、c1t5 という 5 つの変数を参照しています。

```
array c1t{5};
```

配列を参照する場合の規則

配列を参照する場合は、DATA ステップ内で、配列参照(Array Reference)ステートメントより前に ARRAY ステートメントを記述して配列を定義しておく必要があります。配列の定義後は、次のことが行えます。

- SAS 式を記述できるどの位置でも、配列参照ステートメントを使用できます。
- 一部の SAS 関数で、配列参照を引数として使用できます。
- 大かっこ[ ]、中かっこ{ }、かっこ( )で囲んだ添字を使用して配列を参照できます。
- 特殊な配列添字であるアスタリスク(*)を使用することで、INPUT ステートメントや PUT ステートメント内の配列にある変数をすべて参照したり、関数の引数に含まれている変数をすべて参照したりできます。

注: アスタリスク(*)は_TEMPORARY_による一時的な配列では使用できません。

定義した配列は、定義した DATA ステップの処理内でのみ有効です。同一の配列をいくつかの DATA ステップで使用したい場合は、各 DATA ステップ内で配列を再定義しなければなりません。ただし、後続の DATA ステップで同じ変数を持つ配列を再定義する場合は、マクロ変数を使用できます。マクロ変数は、必要な変数名をあらかじめ格納しておく場合に便利です。次に例を示します。

```
%let list=NC SC GA VA;

data one;
   array state{*} &list;
   … more SAS statements …
run;

data two;
   array state{*} &list;
   … more SAS statements …
run;
```

基本的な配列処理の応用

配列の要素数の効率的な判定

反復 DO ステートメントの中で DIM 関数を使用すると、1 次元配列に含まれる要素の数、または多次元配列内の指定した次元に含まれる要素の数が返されます(次元の下限が 1 のとき)。DIM 関数を使用すると、配列要素の数を変更するたびに反復 DO グループの上限を変更しなくても済むようになります。

DIM 関数の形式は、次のとおりです。

DIM*n*(*array-name*)

ここで、*n* には次元を指定します。デフォルト値は 1 です。

また、DIM 関数は、アスタリスク(*)で定義された配列の要素数を引用するときにも使用できます。DIM 関数の例を次に示します。

- **do i=1 to dim(days);**
- **do i=1 to dim4(days) by 2;**

DO WHILE 式と DO UNTIL 式

配列の処理でよく利用されるのは、配列参照を DO WHILE 式または DO UNTIL 式の中で使用する反復 DO ループです。この例では、Trend という配列の要素を反復 DO ループを利用して処理しています。

```
data test;
   array trend{5} x1-x5;
   input x1-x5 y;
   do i=1 to 5 while(trend{i}<y);
   … more SAS statements …
   end;
   datalines;
… data lines …
```

```
;
```

変数リストを使用した配列の簡易定義

SAS System では、次の 3 つの変数リスト名が予約されています。

- _CHARACTER_
- _NUMERIC_
- _ALL_

これらの変数リスト名を使用して、同一の DATA ステップ内ですでに定義されている変数を参照できます。変数リスト_CHARACTER_では、文字変数のみを参照できます。変数リスト_NUMERIC_では、数値変数のみを参照できます。変数リスト_ALL_では、変数がどのように定義されているかに応じて、すべての SAS 変数を参照できます。

次の INPUT ステートメントの例では、$8.入力形式を使用して変数 X1 - X3 を文字値として読み込み、変数 X4 と X5 を数値として読み込んでいます。その次の ARRAY ステートメントでは、変数リスト_CHARACTER_を使用して、配列に含まれる文字変数のみを格納しています。アスタリスク(*)を指定すると、SAS System は、配列内の変数の数をカウントすることにより配列の要素数を判定します。

```
input (X1-X3) ($8.) X4-X5;
array item {*} _character_;
```

変数リスト_NUMERIC_は、通貨を変換する場合などのアプリケーションで使用できます。このプログラムでは、個々の変数名を指定するのではなく、アスタリスクを使用してすべての値を新しい通貨へと変換しています。

変数リストの詳細については、“ARRAY Statement” (*SAS Statements: Reference*)を参照してください。

多次元配列:作成と処理

多次元配列の作成

多次元配列を作成するには、各次元で、配列名の後に{*n*, … }という形式で要素の数を記述します。ここで、*n* は多次元配列の次元ごとに記述する必要があります。

2 次元配列の場合、一番右の次元は列に対応し、左の次元は行に対応します。位置が左にある次元ほど、より高い次元を表します。次の ARRAY ステートメントでは、2 つの行と 5 つの列を持つ 2 次元配列を定義しています。この配列には、10 個の変数が含まれています。2 つの都市(c1 と c2)において 5 回測定した気温(t1 - t5)です。

```
array temprg{2,5} c1t1-c1t5 c2t1-c2t5;
```

SAS System では、行を順序どおりに埋めていく方法で多次元配列に変数を配置します。開始位置は配列の左上隅です。変数は、次のような順序で配置されます。

```
c1t1 c1t2 c1t3 c1t4 c1t5
c2t1 c2t2 c2t3 c2t4 c2t5
```

多次元配列を定義したら、配列参照を使用して配列の要素を参照するときは、配列名と添字を使用できます。次の表は、上記の例に対応する配列参照の一部です。

表 23.3 配列 TEMPRG の配列参照

変数	配列参照
c1t1	temprg{1,1}
c1t2	temprg{1,2}
c2t2	temprg{2,2}
c2t5	temprg{2,5}

ネストした DO ループの使用

多次元配列は、通常、ネストした DO ループ内で処理されます。次に示すのは、2 次元配列を処理する形式の例です。

DO *index-variable-1*=1 **TO** *number-of-rows*;
  DO *index-variable-2*=1 **TO** *number-of-columns*;
    ...more SAS statements ...
  END;
END;

配列参照では、複数のインデックス変数を添字として使用することで、配列にある複数の次元を参照できます。次の形式で指定します。

array-name {*index-variable-1, …,index-variable-n*}

次の例では、10 個の変数を持つ配列を作成しています。変数は 2 つの都市(c1 と c2)で測定した 5 回の気温(t1 - t5)です。DATA ステップには 2 つの DO ループが含まれています。

- 外側の DO ループ(DO I=1 TO 2)では内側の DO ループを 2 回処理します。
- 内側の DO ループ(DO J=1 TO 5)では、1 行に含まれる変数すべてに ROUND 関数を適用して、整数に値を丸める処理をします。

DO ループを 1 回反復するたびに、I と J の値に対応する配列要素の値を置き換えています。

```
options linesize=80 pagesize=60;

data temps;
   array temprg{2,5} c1t1-c1t5 c2t1-c2t5;
   input c1t1-c1t5 /
         c2t1-c2t5;
   do i=1 to 2;
     do j=1 to 5;
       temprg{i,j}=round(temprg{i,j});
     end;
   end;
   datalines;
89.5 65.4 75.3 77.7 89.3
73.7 87.3 89.9 98.2 35.6
75.8 82.1 98.2 93.5 67.7
101.3 86.5 59.2 35.6 75.7
```

```
;

proc print data=temps;
   title 'Temperature Measures for Two Cities';
run;
```

作成したデータセット Temps には、整数に丸められた変数の値が保持されています。

アウトプット 23.2 *多次元配列による出力結果*

Temperature Measures for Two Cities

Obs	c1t1	c1t2	c1t3	c1t4	c1t5	c2t1	c2t2	c2t3	c2t4	c2t5	i	j
1	90	65	75	78	89	74	87	90	98	36	3	6
2	76	82	98	94	68	101	87	59	36	76	3	6

上記の例では、DIM 関数を使用しても同じ結果を得ることができます。

```
do
i=1 to dim1(temprg);
   do j=1 to dim2(temprg);
      temprg{i,j}=round(temprg{i,j});
   end;
end;
```

DIM1(TEMPRG)の値は 2 で、DIM2(TEMPRG)の値は 5 です。

配列の範囲の指定

上限と下限の指定

多くの明示的な配列では、配列の各次元の subscript は 1 から *n* までになります。ここで、*n* はこの次元の要素数になります。したがって、配列のその次元では下限が 1 に、上限が *n* になります。たとえば、次の配列の下限は 1 に、上限は 4 になります。

```
array new{4} Jackson Poulenc Andrew Parson;
```

次の ARRAY ステートメントでは、最初の次元の範囲は 1 - 2、2 番目の次元の範囲は 1 - 5 です。

```
array test{2,5} test1-test10;
```

配列次元の範囲を指定する場合の形式は、次のようになります。

{<lower-1:>upper-1<,…<lower-n:>upper-n>}

したがって、上記の ARRAY ステートメントは次のように記述することもできます。

```
array new{1:4} Jackson Poulenc Andrew Parson;
array test{1:2,1:5} test1-test10;
```

配列の下限は、通常は 1 です。通常は、下限を指定する必要はありません。ただし、配列の開始次元が 1 ではない場合には、下限と上限の両方を指定すると便利です。

次の例では、10 個の変数が Year76 - Year85 と命名されています。ARRAY ステートメントは、変数を First と Second という 2 つの配列に配置しています。

```
array first{10} Year76-Year85;
array second{76:85} Year76-Year85;
```

最初の ARRAY ステートメントでは、要素 first{4}は変数 Year79、first{7}は Year82 になります。他の配列要素も同様に配置されます。2 番目の ARRAY ステートメントでは、配列 second{79}は Year79、second{82}は Year82 になります。

配列 Second を DO グループで処理するには、次のようにして、DO ループの範囲を配列の範囲と一致させます。

```
do i=76 to 85;
   if second{i}=9 then second{i}=.;
end;
```

配列の範囲の判定:LBOUND 関数と HBOUND 関数

LBOUND 関数と HBOUND 関数を使用すると、配列の範囲を判定できます。LBOUND 関数は、1 次元配列の下限、または多次元配列の指定次元の下限を返します。HBOUND 関数は、1 次元配列の上限、または多次元配列の指定次元の上限を返します。

LBOUND 関数と HBOUND 関数の形式は、次のとおりです。

LBOUND*n*(*array-name*)

HBOUND*n*(*array-name*)

n
次元を指定します。デフォルト値は 1 です。

LBOUND 関数と HBOUND 関数を使用すると、反復 DO ループの開始値と終了値を指定して、配列 Second の要素を処理することができます。

```
do i=lbound{second} to hbound{second};
   if second{i}=9 then second{i}=.;
end;
```

この例では、反復 DO ステートメント内にあるインデックス変数の範囲は 76 から 85 までになります。

DIM 関数のかわりに HBOUND 関数を使用する場合

次の ARRAY ステートメントは、72 を下限、76 を上限とする 5 つの要素を含んだ配列を定義します。これらの配列要素は、カレンダ年の 1972 - 1976 年を表しています。

```
array years{72:76} first second third fourth fifth;
```

配列 YEARS を反復 DO ループで処理するには、次のように、DO ループの範囲を配列の範囲と一致させます。

```
do i=lbound(years) to hbound(years);
   if years{i}=99 then years{i}=.;
end;
```

LBOUND(YEARS)の値は 72 で、HBOUND(YEARS)の値は 76 です。

この例では、DIM 関数を使用すると、配列 YEARS の要素数である 5 という値が返されます。したがって、HBOUND 関数ではなく DIM 関数を使用してこの配列の上限を求めた場合、DO ループ内部のステートメントは実行されません。

2 次元配列での範囲の指定

次のリストには、X60 から X99 まで 40 個の変数が挙げられています。これらの変数は、それぞれ 1960 - 1999 年を表します。

```
X60   X61   X62   X63   X64   X65   X66   X67   X68   X69
X70   X71   X72   X73   X74   X75   X76   X77   X78   X79
X80   X81   X82   X83   X84   X85   X86   X87   X88   X89
X90   X91   X92   X93   X94   X95   X96   X97   X98   X99
```

次の ARRAY ステートメントは、配列内の変数を 10 年ごとに分けて配置しています。行の範囲を 6 - 9 に、列の範囲を 0 - 9 に指定します。

```
array X{6:9,0:9} X60-X99;
```

配列 X では、変数 X63 は要素 X{6,3}で、変数 X89 は要素 X{8,9}で表されます。配列 X を反復 DO ループで処理するには、次のいずれかの方法を利用します。

- 方法 1:

```
do i=6 to 9;
   do j=0 to 9;
      if X{i,j}=0 then X{i,j}=.;
   end;
end;
```

- 方法 2:

```
do i=lbound1(X) to hbound1(X);
   do j=lbound2(X) to hbound2(X);
      if X{i,j}=0 then X{i,j}=.;
   end;
end;
```

どちらの例でも、変数 X60 - X99 の値が 0 の場合はすべて欠損値に変更しています。方法 1 では、DO ループの範囲を明示的に設定しています。方法 2 では、LBOUND 関数と HBOUND 関数を使用して配列の各次元の範囲を設定しています。

配列処理の例

例 1:文字変数を使用する配列

ARRAY ステートメントでは、文字変数や、文字変数の長さを指定できます。次の例では、NAMES と CAPITALS という 2 つの文字変数の配列を作成しています。ドル記号($)は、要素を文字変数として定義します。すでに文字変数として宣言されている場合には、ドル記号($)は不要です。INPUT ステートメントを使用して、配列 NAMES に含まれる変数をすべて読み込みます。

DO ループの内側にあるステートメントでは、配列 NAMES にある変数の値を UPCASE 関数を使用して大文字に変更します。その後、その大文字の値を配列 CAPITALS の変数に格納します。

```
options linesize=80 pagesize=60;

data text;
   array names{*} $ n1-n5;
   array capitals{*} $ c1-c5;
```

```
   input names{*};
      do i=1 to 5;
         capitals{i}=upcase(names{i});
      end;
   datalines;
smithers michaels gonzalez hurth frank
;

proc print data=text;
   title 'Names Changed from Lowercase to Uppercase';
run;
```

次の出力結果は、作成したデータセット TEXT を示しています。

アウトプット 23.3 *文字変数を使用する配列*

Names Changed from Lowercase to Uppercase

Obs	n1	n2	n3	n4	n5	c1	c2	c3	c4	c5	i
1	smithers	michaels	gonzalez	hurth	frank	SMITHERS	MICHAELS	GONZALEZ	HURTH	FRANK	6

例 2:配列要素への初期値の割り当て

この例では、配列 Test に変数を作成し、90、80、70 という初期値を割り当てます。値を Score という別の配列に読み込み、Score の各要素を、Test 内にある対応する要素と比較します。Score に含まれる要素の値が Test に含まれる要素の値以上である場合は、Score に含まれる要素の値を変数 NewScore に割り当てます。OUTPUT ステートメントを使用して、SAS データセットにオブザベーションを書き出します。

INPUT ステートメントでは、ID という変数の値を読み込み、次に配列 Score に含まれるすべての変数の値を読み込んでいます。

```
options linesize=80 pagesize=60;

data score1(drop=i);
   array test{3} t1-t3 (90 80 70);
   array score{3} s1-s3;
   input id score{*};
   do i=1 to 3;
      if score{i}>=test{i} then
         do;
            NewScore=score{i};
            output;
         end;
   end;
   datalines;
1234  99 60 82
5678  80 85 75
;

proc print noobs data=score1;
   title 'Data Set SCORE1';
run;
```

次の出力結果は、作成したデータセット Score1 を示しています。

アウトプット 23.4 *配列要素への初期値の割り当て*

Data Set SCORE1

t1	t2	t3	s1	s2	s3	id	NewScore
90	80	70	99	60	82	1234	99
90	80	70	99	60	82	1234	82
90	80	70	80	85	75	5678	85
90	80	70	80	85	75	5678	75

例 3:現在の DATA ステップにおいて一時的に使用する配列の作成

配列の要素が、DATA ステップの処理中のみ必要になる定数である場合は、配列グループに変数を指定せずに、一時的な配列要素をかわりに使用することもできます。一時的な配列は、配列名と次元を指定することで参照します。一時的な配列は、変数と同様に機能します。ただし、変数名を持たず、出力データセットにも格納されません。一時的な配列は自動的に保持され、次の DATA ステップ反復が開始されても欠損値にはリセットされません。

一時的な配列を作成するには、変数リスト_TEMPORARY_を使用します。次の例では、Test という一時的な配列を作成しています。

```
options linesize=80 pagesize=60;

data score2(drop=i);
      array test{3} _temporary_ (90 80 70);
      array score{3} s1-s3;
      input id score{*};
         do i=1 to 3;
            if score{i}>=test{i} then
               do;
                  NewScore=score{i};
                  output;
               end;
         end;
      datalines;
   1234  99 60 82
   5678  80 85 75
   ;

   proc print noobs data=score2;
      title 'Data Set SCORE2';
   run;
```

次の出力結果は、作成したデータセット Score2 を示しています。

アウトプット 23.5 _TEMPORARY_配列を使用した出力結果

Data Set SCORE2

s1	s2	s3	id	NewScore
99	60	82	1234	99
99	60	82	1234	82
80	85	75	5678	85
80	85	75	5678	75

例 4: すべての数値変数に対する操作の実行

この例では、配列 Test に含まれるすべての数値変数に 3 を掛けます。

```
options nodate pageno=1 linesize=80 pagesize=60;

data sales;
   infile datalines;
   input Value1 Value2 Value3 Value4;
   datalines;
11 56 58 61
22 51 57 61
22 49 53 58
;
data convert(drop=i);
   set sales;
   array test{*} _numeric_;
   do i=1 to dim(test);
      test{i} = (test{i}*3);
   end;
run;

proc print data=convert;
   title 'Data Set CONVERT';
run;
```

次の出力結果は、作成したデータセット CONVERT を示しています。

アウトプット 23.6 _NUMERIC_変数リストを使用した出力結果

Data Set CONVERT

Obs	Value1	Value2	Value3	Value4
1	33	168	174	183
2	66	153	171	183
3	66	147	159	174

4 部

SAS ファイルの概念

24 章

SAS ライブラリ

SAS ライブラリの定義

SAS ライブラリの論理的構造は、動作環境にかかわらず共通です。SAS System のインストールが可能な動作環境であれば、SAS ファイルを編成、配置、管理するための構造は同じです。

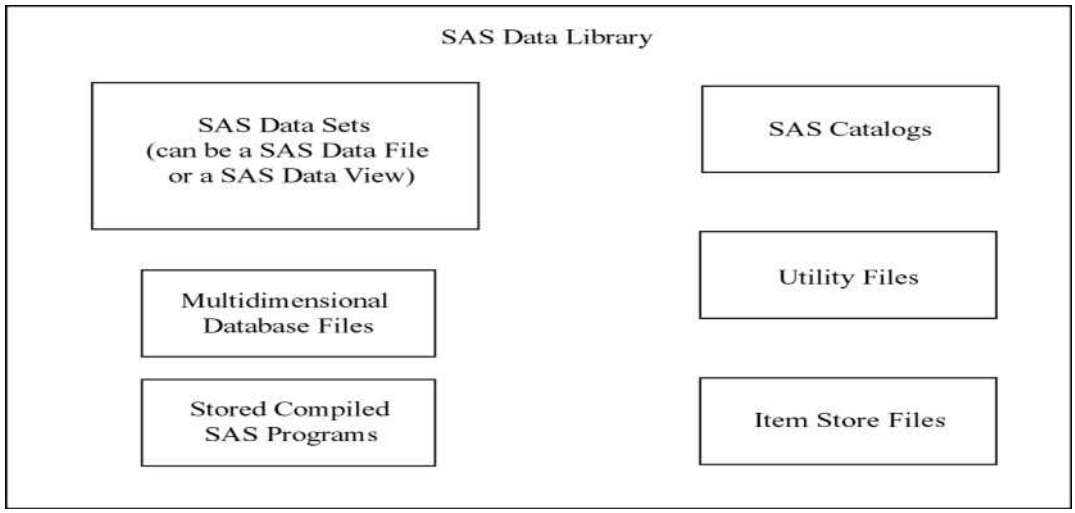
ただし、SAS ライブラリの物理的な実装形式は、動作環境によって異なります。ほとんどの SAS ライブラリは、動作環境がファイルを格納したりアクセスしたりするのと同様の方法で、SAS ファイルの格納を実装します。

たとえば、ディレクトリによってファイルを管理する動作環境では、SAS ライブラリとは、同一ディレクトリに格納され、同一エンジンによってアクセスされる SAS ファイルの集まりを意味します。それ以外のファイルをそのディレクトリに格納することはできますが、SAS System によって割り当てられたファイル拡張子を持つファイルだけが、SAS ライブラリのメンバとして認識されます。z/OS 環境では、SAS ライブラリを、従来の OS データセット内に SAS ファイルのみを含む連結ライブラリとして、または UNIX システムサービスでのディレクトリとして実装することができます。

SAS ファイルには、次の種類があります。

- SAS データセット(SAS データファイルまたは SAS ビュー)
- SAS カタログ
- コンパイル済みストアド SAS プログラム
- SAS Utility ファイル
- ACCESS ディスクリプタ
- MDDB ファイル、FDB ファイル、DMDB ファイルなどの多次元データベースファイル
- アイテムストアファイル(予備ファイル)

図 24.1 *SAS ライブラリのファイルの種類*



SAS ファイルは、それぞれ固有の形式で情報を格納します。これらの形式は、**SAS ファイルタイプ**によって異なります。代表的な SAS ファイルには、SAS データファイルと SAS カタログがあります。SAS データファイルは、変数をオブザベーションによる表形式でデータ値を格納します。SAS カタログは、**エントリ**という形式でさまざまな情報を格納します。SAS System は、ファイルの作成や指定を行う SAS プログラムに基づいてフ

ァイルの種類を確定します。このため、SAS ライブラリには、同一の名前でメンバタイプが異なる SAS ファイルを格納することができます。

SAS ライブラリには、ユーザーが作成した SAS ファイルを格納できます。また、SAS セッションの開始時に SAS System が自動的に定義する Work ライブラリなどの特別なライブラリもあります。1 つの SAS ライブラリに格納できる SAS ファイルの数には、上限はありません。

ライブラリエンジン

各 SAS ライブラリには、それぞれ 1 つの SAS ライブラリエンジンが割り当てられます。SAS ライブラリエンジンは、SAS System と SAS ライブラリとの間の入出力インターフェイスを形成するソフトウェアコンポーネントです。SAS ライブラリエンジンは、SAS ライブラリ内の SAS ファイルを検索し、SAS ファイルの内容を SAS System に認識可能な形式で渡します。ライブラリエンジンは、次のようなタスクを実行します。

- データの読み込みおよび書き出し
- SAS データのライブラリ内の SAS ファイルの一覧の取得
- SAS ファイルの削除および名前の変更

SAS System は複数のライブラリエンジンを経由して SAS ファイルにアクセスできるマルチエンジンアーキテクチャ(MEA)を採用しているため、次のようなさまざまな形式の SAS ファイルを読み書きできます。それぞれの SAS エンジンには固有の処理機能があります。

- 以前のバージョンの SAS System で生成された SAS ファイルの処理
- SAS System 以外のソフトウェアプログラムで作成されたデータベースファイルの読み込み
- ディスクやテープへの SAS ファイルの格納およびアクセス
- SAS ファイル内の変数およびオブザベーションの配置の確定
- データの物理的な領域からメモリへの展開
- 異なる動作環境間での SAS ファイルの移送

一般に、エンジンがデータを処理している間は、処理をしているエンジンの種類についてユーザーが意識する必要はありません。ユーザーが発行した命令がエンジンでサポートされていない場合は、エラーメッセージが SAS ログに表示されます。このような場合は、特定の処理タスクを実行するエンジンを必要に応じて選択できます。ただし、通常は、SAS System が自動的に適切なエンジンを選択するため、ユーザーがエンジンを指定する必要はありません。

DATA ステップを処理するときに、複数のエンジンを必要とする場合があります。たとえば、あるエンジンを使用してデータを入力し、別のエンジンを使用してオブザベーションを出力データセットに書き出す場合などです。

ライブラリエンジンの詳細や、Base SAS ソフトウェアで利用可能なエンジンの一覧については、“ライブラリエンジンについて” (742 ページ)を参照してください。

ライブラリ名

物理名と論理名(ライブラリ参照名)

SAS ライブラリを使用するには、その SAS ライブラリがどこに置かれているかを SAS System に伝える必要があります。SAS System は、動作環境または SAS System の命名規則に基づいて、SAS ライブラリを認識します。SAS ライブラリの定義には、次の 2 種類があります。

- 動作環境が認識するファイル保存場所に割り当てる物理名
- LIBNAME ステートメント、LIBNAME 関数、**ライブラリ作成**ウィンドウのいずれかを使用してライブラリ参照名として割り当てる論理名

SAS ライブラリの物理名とは、動作環境が SAS ファイルの物理的な保存場所を認識するための名前です。物理名は、動作環境の命名規則に従う必要があります。動作環境は物理名によって、ディレクトリまたは SAS ライブラリが含むデータセットを識別します。

論理名、すなわちライブラリ参照名とは、SAS System が SAS ライブラリの物理名を認識する名前です。ライブラリ参照名は、各 SAS ジョブまたは SAS セッションにおいて、SAS ライブラリの物理名に割り当てます。

ライブラリ参照名の割り当て

ライブラリ参照名の割り当てには、次の方法を使用できます。

- LIBNAME ステートメント
- LIBNAME 関数
- **ライブラリ作成**ウィンドウは、ツールバーからも表示できます。
- 動作環境コマンド

ライブラリ参照名を割り当てると、SAS ライブラリ内のファイルの読み込み、作成、更新を行うことができます。ライブラリ参照名は現在の SAS セッションでのみ有効です。ただし、ライブラリ参照名を割り当てるときに**ライブラリ作成**ウィンドウを使用し、**起動時に有効**チェックボックスを選択した場合は、次の SAS セッションでも有効となります。

ライブラリ参照名の最大の長さは、8 文字です。プログラム中でライブラリ参照名が割り当てられていることを確認するには、LIBREF 関数を使用します。SAS セッション内では、ライブラリ参照名を繰り返し参照できます。SAS セッションで割り当てることができるライブラリ参照名の数には、上限はありません。ただし、動作環境またはサイトにより上限が設定されている場合があります。バッチモードで実行している場合、ライブラリ参照名を割り当てるには、あらかじめそのライブラリが存在している必要があります。対話型モードでは、ライブラリが存在しない場合にはライブラリを作成できます。

動作環境の情報
: 各動作環境での LIBNAME ステートメントの使用例を次に示します。ライブラリ参照名の割り当ておよび使用に関するルールは、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

表 24.1 ライブラリ参照名を割り当てるための構文

動作環境	例
DOS、Windows	`libname mylibref 'c:\root\mystuff\sasstuff\work';`
UNIX	`libname mylibref '/u/mystuff/sastuff/work';`
z/OS	`libname mylibref 'userid.mystuff.sastuff.work';` `libname mylibref '/mystuff/sastuff/work';`

ライブラリ参照名を使用しないで SAS ファイルにアクセスすることもできます。“ライブラリ参照名を使用せずに永久 SAS ファイルにアクセスする” (579 ページ).を参照してください。

LIBNAME ステートメントを使用した論理名(ライブラリ参照名)の割り当てと割り当て解除

LIBNAME ステートメントまたは LIBNAME 関数を使用すると、物理名に対してライブラリ参照名を割り当てることや、そのような割り当てを解除することができます。LIBNAME ステートメントの詳細については *SAS Statements: Reference* を、LIBNAME 関数の詳細については *SAS Functions and CALL Routines: Reference* を参照してください。

動作環境の情報
: 一部の動作環境では、動作環境のコマンドを使用して、SAS ライブラリにライブラリ参照名を割り当てることができます。動作環境のコマンドを使用して、SAS ライブラリにライブラリ参照名を割り当てると、ライブラリ参照名を作成した SAS セッションが終了した後も、割り当てが有効な場合があります。また、LIBNAME ステートメントまたは LIBNAME 関数しか使用できない動作環境もあります。ライブラリ参照名の割り当ての詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

次の例では、LIBNAME ステートメントの最も一般的な形式を使用して、SAS ライブラリの物理名にライブラリ参照名 Annual を割り当てます。

```
libname annual 'SAS-library';
```

LIBNAME ステートメントを使用してライブラリ参照名を割り当てた場合は、各 SAS セッションの終了時に、ライブラリ参照名が自動的に解除(割り当ての取り消し)されます。セッションが終了する前にライブラリ参照名 Annual を解除する場合は、次の形式の LIBNAME ステートメントを発行します。

```
libname annual clear;
```

また、**ライブラリの作成**ウィンドウを使用したライブラリ参照名の割り当てまたは解除の実行や、エクスプローラウィンドウを使用した SAS ライブラリの表示、追加、削除が行えます。これらのウィンドウを表示する**ライブラリの作成**または **SAS エクスプローラ**アイコンは、ツールバー上にあります。

予約済みライブラリ参照名

SAS System では、特定の用途のためにいくつかの SAS 名が予約されています。Sashelp、Sasuser、Work は、システムが使用するライブラリ参照名として予約されているため、ライブラリ参照名には使用できません。これらのライブラリの目的と内容については、“永久ライブラリと一時ライブラリ” (573 ページ)を参照してください。

動作環境の情報
一部の動作環境では、他にも SAS System 用に予約されたライブラリ参照名があります。また、動作環境で予約されている特定のキーワードをライブラリ参照名として使用できない場合もあります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

SAS/CONNECT サーバー、SAS/SHARE サーバー、WebDAV サーバー上のリモート SAS ライブラリへのアクセス

SAS/CONNECT と SAS/SHARE のリモートライブラリアクセス

LIBNAME ステートメントを使用すると、サーバー(リモート)データの読み取り、書き出し、更新が、まるで同データがクライアントのディスク上に保存されているかのように実行できます。SAS System は、クライアントメモリ内のデータを処理します。このデータは、サーバーデータに対する後続するクライアント要求があると上書きされます。

LIBNAME ステートメントを使うことで、異なるアーキテクチャを持つコンピュータ上にある SAS データセットにアクセスできます。また、LIBNAME ステートメントは、異なるアーキテクチャを持つコンピュータ上にある一部の SAS カタログエントリタイプに対しては読み取り専用アクセスを提供します。

LIBNAME ステートメントは、SAS ライブラリ参照名と永久 SAS ライブラリを関連付けることにより、リモートサーバーへのアクセスを提供します。

SAS/CONNECT の例:

次の例では、SAS/CONNECT クライアントが、SAS/CONNECT サーバー上にあるサーバーライブラリにアクセスする LIBNAME ステートメントを生成します。クライアントは、新しいライブラリ参照名 Reports を作成します。

```
signon rempc;
libname reports 'd:\prod\reports' server=rempc;
```

上記の例では、SAS/CONNECT クライアントは、REMPC という名前の SAS/CONNECT サーバーにサインオンします。この結果、サーバーライブラリがクライアントセッションに割り当てられます。SERVER=の値には、SIGNON ステートメントで使用したサーバーセッション ID と同じ値を指定します。

SAS/ACCESS ソフトウェアの機能に関する詳細については、*SAS/CONNECT User's Guide* を参照してください。

SAS/SHARE の例:

次の例では、SAS/SHARE クライアントは LIBNAME ステートメントを使用して、既存のライブラリ参照名 Sales でサーバーライブラリにアクセスします。このライブラリ参照名 Sales は、クライアントのアクセス用に SAS/SHARE サーバーで事前定義されたものです。

```
libname sales server=server1;
```

SAS/SHARE ソフトウェアの機能に関する詳細については、*SAS/SHARE User's Guide* を参照してください。

WebDAV サーバーのリモートライブラリアクセス

WebDAV (Web Distributed Authoring and Versioning)は、HTTP の拡張プロトコルです。WebDAV は、インターネットを通じた共同オーサリングのための標準インフラストラクチャです。WebDAV を使うと、Web ドキュメントの編集、各バージョンを保存し後で取り出せるようにすること、上書きを禁止するためのロック機構の提供などが行えます。SAS System は、UNIX および Windows 動作環境において WebDAV プロトコルをサポートしています。

LIBNAME ステートメントを使用して WebDAV サーバーにアクセスする例を次に示します。

```
libname davdata v9 "http://www.webserver.com/users/mydir/datadir"
       webdav user="mydir" pw="12345";
```

WebDAV 上のファイルにアクセスすると、SAS System はそのファイルを同サーバーからユーザーのローカルディスクへとコピーし、同ファイルを処理できるようにします。これらのファイルは SAS Work ディレクトリに一時的に格納されます。一時的な格納場所に別のディレクトリを指定するには、LIBNAME ステートメントの LOCALCACHE= オプションを使用します。同ファイルの更新が完了すると、SAS System は当該ファイルをそれが保存されていた WebDAV サーバーへ戻し、同ファイルをローカルディスクから削除します。

詳細については、“WHEREUP= Data Set Option” (*SAS Data Set Options: Reference*)を参照してください。

ライブラリ連結

ライブラリ連結の定義

ライブラリの連結とは、複数のライブラリを論理的に結合することです。ライブラリの連結を使用することにより、複数のライブラリが保持する SAS データセットに、1 つのライブラリ参照名を使用してアクセスできます。

複数の SAS ライブラリを連結するには、それらのライブラリ参照名または物理名を LIBNAME ステートメントまたは LIBNAME 関数に指定します。

物理名は、一重引用符(')または二重引用符(")で囲んで LIBNAME ステートメントに指定します。引用符で囲まれていない名前がある場合、SAS System は、すでに割り当てられているライブラリ参照名で同じ名前を検索します。

次に示す連結ライブラリの例では、Summer、Winter、Spring、Fall、Annual は、すでに定義されているライブラリ参照名です。

```
libname annual (summer winter spring fall);

libname annual ('SAS-library-1' 'SAS-library-2'  'SAS-library-3');

libname annual ('SAS-library' winter spring fall);

libname total (annual 'SAS-library');
```

SAS でのライブラリメンバの連結方法

同じ名前のメンバが複数のライブラリに存在する場合は、最初に検出されたメンバが、入力および更新処理の対象として使用されます。出力は常に最初のライブラリに対して行われます。

次の例には、3 つの SAS ライブラリがあり、各ライブラリにはそれぞれ 2 つの SAS データファイルが含まれています。

Lib1
    Apples および Pears

Lib2
    Apples および Oranges

Lib3
Oranges および Plums

LIBNAME ステートメントは、Lib1、Lib2、Lib3 を次のように連結します。

```
libname fruit (lib1 lib2 lib3);
```

連結ライブラリ Fruit には、次のメンバが含まれます。

- Apples
- Pears
- Oranges
- Plums

注: 出力は常に最初のライブラリに対して行われます。たとえば、次のステートメントにより作成されたデータファイルは、ライブラリ Lib1 に書き出されます。

```
data fruit.oranges;
```

この例では、ライブラリ Lib1 内のデータファイル Apples がライブラリ Lib2 内のデータファイル Apples と異なっている場合に、Apples の更新が指定されたならば、ライブラリ Lib1 内のデータファイルのみが更新されます。これは、最初に検出されたメンバ Apples がライブラリ Lib1 のデータファイルであるためです。

連結ライブラリの詳細については、"LIBNAME Statement" (*SAS Statements: Reference*)を参照してください。

動作環境の情報
動作環境固有の連結方法の詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

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任意のコンテキストにおいて、その連結ライブラリのライブラリ参照名を指定できるようになります。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す。

- オプションまたはエンジンを指定する場合は、物理名で指定されたライブラリにのみ適用されます。ライブラリ参照名で指定された論理名のライブラリには適用されません。
- 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。
- 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 ファイルの一覧を出力する場合は、同一のファイル名は常に 1 回のみ表示されます。連結ライブラリの中にファイル名が複数回出現する場合でも、表示されるのは最初に検出されたファイル名のみです。たとえば、ライブラリ One に A.DATA が含まれており、ライブラリ Two にも A.DATA が含まれている場合は、最初に検出されたファイル名である、ライブラリ One の A.DATA のみが表示されます。

  別のファイルに論理的に関連付けられている SAS ファイル(データファイルのインデックスなど)が一覧に出力されるのは、その親ファイルが当該ファイル名の最初のオカレンスとなる場合のみです。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。ライブラリ Two の A.DATA および A.INDEX はリストされません。

- 連結ライブラリにシーケンシャルライブラリが 1 つでも存在する場合、その連結ライブラリは、ランダムアクセスを必要とするアプリケーションによってシーケンシャルファイルであると見なされます。たとえば、DATASETS プロシジャはシーケンシャルライブラリを処理できないため、ある連結ライブラリが 1 つまたは複数のシーケンシャルライブラリを含んでいる場合、DATASETS プロシジャはその連結ライブラリを処理できません。
- 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。
- ライブラリ参照名に連結ライブラリを指定した場合は、そのライブラリ参照名を変更しても、連結には影響しません。
- 連結ライブラリ内のデータセット名を既存のデータセット名に変更することはできません。

永久ライブラリと一時ライブラリ

SAS ライブラリは、通常、永久データライブラリとして保存されます。ただし、SAS セッションまたは SAS ジョブの間だけ SAS ファイルを格納する一時ライブラリを使用することもできます。

永久ライブラリとは、コンピュータのディスク領域に存在するライブラリであり、SAS セッションが終了しても削除されません。永久ライブラリは、ユーザーが明示的に削除するまで保持されます。永久ライブラリは、後続の SAS セッションの処理で利用できます。永久ライブラリの SAS ファイルで作業する場合は、通常、ライブラリ参照名を含む 2 レベル名を使用します。SAS System はライブラリ参照名によって、SAS ファイルを検索または格納する場所を認識します。

注: オペレーティングシステムで認識できる構文を使用すると、ライブラリ参照名の使用を省略して、使用するファイルを直接指定することもできます。Windows 環境の例を次に示します。

```
data 'C:\root\sasfiles\myfile.ext';
```

動作環境の情報
ファイルの指定方法は、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

一時 SAS ライブラリとは、現在の SAS セッションまたは SAS ジョブの間だけ存在するライブラリです。SAS セッションまたは SAS ジョブの間に作成される SAS ファイルは、一時的に作成される作業領域に保持されます。この作業領域は、ディスク領域である場合とそうでない場合とがあります。この作業領域には、通常、デフォルトのライブラリ参照名 Work が割り当てられます。SAS セッションの間、一時ライブラリ Work にあるファイルは任意の DATA ステップまたは SAS プロシジャで使用できます。ただし、後続の SAS セッションで使用することはできません。Work ライブラリでは、データセットの格納や取得に、通常、1 レベル名を指定します。Work ライブラリに保持されている SAS ファイルは、SAS セッションの正常終了時に削除されます。

永久ライブラリおよび一時ライブラリに対する命名および作業方法をカスタマイズする SAS システムオプションは多数あります。詳細については、*SAS System Options: Reference* にある USER=、WORK=、WORKINIT、WORKTERM のシステムオプションそれぞれの説明を参照してください。

メタデータ連結ライブラリの定義

メタデータ連結ライブラリは、対応するメタデータ保護テーブルオブジェクトに結び付けられる物理的なライブラリです。メタデータ連結ライブラリ内のそれぞれの物理テーブルはそのヘッダーに情報を持ち、ヘッダーが特定のメタデータオブジェクトをポイントしています。そのポインタは、物理的テーブルとメタデータオブジェクト間のセキュリティバインドを形成しています。バインドによって、ユーザーが SAS からのアクセスをどのように要求するかにかかわらず、SAS は物理テーブルのメタデータ層アクセス要件を例外なく確実に適用できます。詳細については、*SAS Guide to Metadata-Bound Libraries* を参照してください。

AUTHLIB プロシジャは、メタデータ連結ライブラリの作成、アクセス、変更に使用されます。このプロシジャの使用は SAS 管理者を対象としています。メタデータ層やホスト層に十分な権限がないユーザーはこのプロシジャを使用できません。詳細については、“AUTHLIB” (*Base SAS Procedures Guide*)を参照してください。

SAS システムライブラリ

SAS システムライブラリの概要

SAS で用意された 4 つの特殊ライブラリでは、利便性、サポート、およびカスタマイズ機能が提供されます。

- Work
- User
- Sashelp
- Sasuser

Work ライブラリ

Work ライブラリの定義

Work ライブラリは、SAS セッションの間に使用される一時ファイルを保持するために、各 SAS セッションの開始時に自動的に定義される一時ライブラリです。Work ライブラリには、2 種類の一時ファイルが格納されます。1 つはユーザーが作成する一時ファイルで、もう 1 つは SAS System が処理の過程で内部的に作成する一時ファイルです。通常、Work ライブラリは、各 SAS セッションの正常終了時に削除されます。

Work ライブラリの使用

Work ライブラリにある SAS ファイルの格納や取得を行うには、SAS プログラムステートメントで 1 レベル名を指定します。これらのファイルには、ライブラリ参照名 Work がシステムデフォルトとして自動的に割り当てられます。ただし、明示的にライブラリ参照名 User を割り当てた場合を除きます。次の例は、Work ライブラリに格納される SAS データセットに有効な名前を含むステートメントです。

- `data test2;`
- `data work.test2;`

- `proc contents data=testdata;`
- `proc contents data=work.testdata;`

動作環境の情報
Work ライブラリの実装形式は、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

User ライブラリとの関係

Work ライブラリが、現在の SAS セッションの間に使用される一時ファイルを保持することを目的としているのに対し、User ライブラリは、SAS セッションが終了した後もファイルを保持することを目的としています。ライブラリ参照名 User を SAS ライブラリに割り当てる場合は、1 レベル名を使用して SAS ファイルの作成およびアクセスができます。これらの SAS ファイルは、SAS セッションの終了時に削除されません。SAS System は 1 レベルのファイル名を検出すると、(User ライブラリが定義されている場合は)最初に User ライブラリを検索し、その後 Work ライブラリを検索します。User ライブラリを定義すると、そこに保存した SAS ファイルは SAS セッション終了時に削除されず、また 1 レベル名のファイルはデフォルトで User ライブラリに置かれるようになります。この場合、Work ライブラリに一時ファイルを作成するには、Work.Name のような 2 レベル名を使用することが必要です。

User ライブラリ

User ライブラリの定義

User ライブラリを使用すると、1 レベル名で Work ライブラリ以外の SAS ライブラリでファイルの読み取り、作成、書き出しができます。ライブラリ参照名を SAS ファイル名の一部として指定する必要はありません。ある SAS ライブラリにライブラリ参照名 User を割り当てると、1 レベル名のファイルはすべて、その SAS ライブラリに格納されます。Work ライブラリとは異なり、この SAS ライブラリに格納されているファイルは、SAS セッションの終了時に削除されません。

User ライブラリ参照名の割り当て方法

User ライブラリ参照名の割り当てには、次のいずれかの方法を使用できます。

- LIBNAME ステートメント
- LIBNAME 関数
- USER=システムオプション
- 動作環境のコマンド

次の例では、DATA ステップで LIBNAME ステートメントを使用しています。この DATA ステップでは、データセット Region を永久 SAS ライブラリに格納します。

```
libname user 'SAS-library';
data region;
... more DATA step statements ...
run;
```

次の例では、LIBNAME 関数を使用して User ライブラリ参照名を割り当てています。

```
data _null_;
   x=libname ('user', 'SAS-library');
run;
```

USER=システムオプションを使用してライブラリ参照名を割り当てる場合は、最初にライブラリ参照名を SAS ライブラリに割り当てる必要があります。次に USER=システムオプションを使用して、そのライブラリを 1 レベル名のためのデフォルトとして指定しま

す。この例で、DATA ステップでは、データセット Prochlor を SAS ライブラリ Testlib に格納します。

```
libname testlib 'SAS-library';
options user=testlib;
data prochlor;
... more DATA step statements ...
run;
```

動作環境の情報

User ライブラリ参照名の割り当て方法とその結果は、動作環境によって多少異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

Work ライブラリとの関係

User ライブラリ参照名が割り当てられると、1 レベル名のためのデフォルトのライブラリ参照名 Work は無効になります。1 レベル名で SAS ファイルを参照すると、最初にライブラリ参照名 User が検索されます。ある SAS ライブラリに User が割り当てられている場合、1 レベル名の SAS ファイルはそのライブラリに格納されます。ライブラリ参照名 User がライブラリに割り当てられていない場合、1 レベル名の SAS ファイルは一時ライブラリ Work に格納されます。User ライブラリ参照名が割り当てられている場合に Work ライブラリの SAS ファイルを参照するには、Work を付けた 2 レベル名をライブラリ参照名として指定する必要があります。SAS System が内部的に作成するデータファイルは、常に Work ライブラリに格納されます。

Sashelp ライブラリ

SAS サイトごとに Sashelp ライブラリがあり、そこには SAS セッションのさまざまな要素を制御するための情報が含まれた、カタログやその他のファイルのグループが含まれています。このライブラリには、すべてのユーザーのデフォルト値として使用する設定値が格納されています。ユーザーの個人用の設定は Sasuser ライブラリに格納されます。Sasuser ライブラリについては、次のセクションで説明します。

Base SAS ソフトウェア以外の SAS プロダクトがインストールされている場合、Sashelp ライブラリには、それらのプロダクトで使用されるカタログが含まれます。多くの場合、Sashelp ライブラリ内のデフォルトの設定は、SAS System 管理者により、サイトに合わせてカスタマイズされています。サイトで格納されているカタログの一覧を出力するには、このセクションで説明されるファイル管理ユーティリティのいずれかを使用します。

Sasuser ライブラリ

Sasuser ライブラリには、SAS の機能を必要に応じてカスタマイズできる、SAS カタログが含まれています。Sashelp ライブラリ内のデフォルト値が使用するアプリケーションに適していない場合は、それらをカスタマイズしたデフォルト設定を Sasuser ライブラリに格納することができます。たとえば、Base SAS ソフトウェアでは、ファンクションキーの設定またはウィンドウの属性に関するユーザー独自の設定を、Sasuser.Profile という個人用のプロファイルカタログに格納できます。

SAS System は、初期化時に、Sasuser システムオプションで指定される情報に応じて、Sasuser ライブラリを割り当てます。

システム管理者は RSASUSER システムオプションを使用して、Sasuser ライブラリへのアクセスモードを制御できます。これにより、1 つの Sasuser ライブラリを複数のユーザーが共有して使用するようにインストールされている場合は、ユーザーによって Sasuser ライブラリが変更されることを防ぐことができます。

動作環境の情報

ほとんどの動作環境では、Sasuser ライブラリが存在していない場合は新規に Sasuser ライブラリが作成されます。ただし、Sasuser ライブラリの実装形式は、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

シーケンシャルデータライブラリ

SAS System には、ディスクやテープのようなシーケンシャルフォーマットデバイス上に格納されているファイルを読み書きするための機能およびプロシジャが用意されています。SAS ライブラリをシーケンシャルフォーマットで格納する前に、次のことを考慮する必要があります。

- ランダムアクセス方式をシーケンシャルライブラリ内の SAS データセットに対して使用することはできません。
- SAS ジョブ内で、シーケンシャルライブラリ内またはテープ上でアクセスできる SAS ファイルは 1 つのみです。

  たとえば、同一のライブラリ内またはテープ上にある複数の SAS データセットを 1 つの DATA ステップで同時に読み込むことはできません。ただし、次のようにアクセスすることは可能です。

  - 異なるシーケンシャルライブラリ内またはテープ上にある複数の SAS ファイルに同時にアクセスできます。これは、複数のテープドライブが利用できる場合に可能です。
  - 同一のシーケンシャルライブラリ内またはテープ上でも、1 つの DATA ステップまたは PROC ステップで 1 つの SAS ファイルにアクセスした後、後続の DATA ステップまたは PROC ステップで別の SAS ファイルにはアクセスできます。

  また、同一の DATA ステップまたは PROC ステップにおいて、テープ上またはシーケンシャルライブラリ内の SAS データセットが複数指定されている場合、コンパイル段階では 1 つの SAS データセットファイルが開かれます。実行段階では別の SAS データセットが開かれます。詳細については、“SET Statement” (*SAS Statements: Reference*)を参照してください。
- 一部の動作環境では、DATA ステップまたは PROC ステップで読み書きができるのは、SAS データセットに限られます。ただし、COPY プロシジャを使用して、SAS ライブラリのすべてのメンバを格納したり、バックアップの目的でテープに転送することは可能です。
- テープを使用するには、サイト固有の注意事項がある場合があります。たとえば、SAS ライブラリを利用する前に、テープを手動でマウントすることが必要な場合があります。テープドライブの使用方法に不明な点がある場合は、SAS System 管理者に問い合わせてください。

シーケンシャルエンジンの詳細については、35 章, “SAS エンジン” (737 ページ)を参照してください。

動作環境の情報

SAS ファイルのシーケンシャルフォーマットでの格納およびアクセスに関する詳細については、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

ライブラリ管理のツール

SAS Utilities

SAS ファイル管理に使用可能な SAS Utilities によって、同一のライブラリに属している複数の SAS ファイルを一度に処理できます。これらのユーティリティを使用することでさまざまな利点が得られます。たとえば、SAS データファイルに割り当てられているインデックスファイル、一貫性制約、監査証跡、バックアップ、世代データセットに対して、コピー、名前の変更、削除を自動的に実行できます。また、SAS ユーティリティプロシジャをさまざまな動作環境で実行することも可能です。

SAS System には、さまざまなファイル管理タスクが可能な SAS ウィンドウ、システムオプション、関数、プロシジャが用意されています。状況に応じて、次の機能を単独で、または組み合わせて使用することができます。SAS ユーティリティプロシジャの詳細については、“Choosing the Right Procedure” (*Base SAS Procedures Guide*)を参照してください。SAS ウィンドウ環境を使用して SAS ファイルを管理する方法については、16 章, “SAS ウィンドウ環境の紹介” (341 ページ) 、17 章, “SAS ウィンドウ環境を用いたデータ管理” (361 ページ) およびオンラインヘルプを参照してください。

CATALOG プロシジャ
: COPY ステートメント、CONTENTS ステートメント、APPEND ステートメントによって、カタログ管理機能を提供します。

DATASETS プロシジャ
: CATALOG を除くすべてのメンバタイプの SAS ファイルに、ライブラリ管理機能を提供します。SAS ウィンドウを使用しない場合や、SAS System をバッチモードまたは対話型ラインモードで実行する場合は、このプロシジャを使用すると、実行時間とリソースを節約できます。

SAS エクスプローラ
: ほとんどのファイル管理タスクを SAS プログラムステートメントをサブミットせずに実行できる各種のウィンドウが含まれています。SAS エクスプローラを使用するには、**ツールバー**ウィンドウに EXPLORER、LIBNAME、CATALOG、DIR のいずれかを入力するか、または**ツールバー**メニューからエクスプローラのアイコンを選択します。

DETAILS システムオプション
: CONTENTS プロシジャまたは DATASETS プロシジャを使用するときの、ファイルに関する情報のデフォルトの表示を設定します。DETAILS システムオプションを有効にすると、使用するプロシジャやウィンドウに応じて、ファイルに関する追加情報を出力します。

ライブラリディレクトリ

SAS エクスプローラおよび SAS プロシジャを使用すると、SAS ライブラリのメンバの一覧、またはディレクトリを取得できます。各ディレクトリには、各メンバ名とメンバタイプが含まれます。メンバタイプが DATA の場合は、インデックス、監査証跡、バックアップ、世代データセットがデータセットに割り当てられているかどうかがディレクトリに表示されます。ディレクトリには、ライブラリの属性の一部も示されます。ただし、表示内容は動作環境によって異なります。

注: SAS ライブラリには、さまざまな SAS ユーティリティファイルが含まれます。これらのファイルはライブラリディレクトリには表示されず、内部処理に使用されます。

ライブラリ参照名を使用せずに永久 SAS ファイルにアクセスする

SAS System で SAS ファイルにアクセスするには、LIBNAME ステートメントでライブラリ参照名を割り当てたり、**ライブラリの作成**ウィンドウを使用したりする以外にも方法があります。この方法を使用するには、ファイル名および SAS ライブラリを一重引用符(')で囲んで、物理名で指定します。

たとえば、ディレクトリベースのシステムで、データセット MyData を、デフォルトのディレクトリ(SAS System を実行しているディレクトリ)に作成する場合は、次のコード行を記述します。

```
data 'mydata';
```

SAS データセットが作成され、SAS セッションを実行している間だけそのデータセットが保持されます。

次のように一重引用符を省略すると、データセット MyData は Work 一時サブディレクトリに作成され、Work.MyData という名前になります。

```
data mydata;
```

データセット MyData を SAS System を実行しているディレクトリ以外のディレクトリに作成する場合は、動作環境の命名規則に従い、パス名全体を一重引用符で囲みます。たとえば、次の DATA ステップは、データセット Foo をディレクトリ **c:\sasrun\mydata** 内に作成します。

```
data 'c:\sasrun\mydata\foo';
```

この方法によるファイルアクセスは、すべての動作環境で機能します。また、*libref.data-set-name* のような 2 レベル名による SAS データセットの指定方法は、ほとんどのプログラムステートメントで動作します。ほとんどのデータセットオプションにおいても、引用符で囲まれた名前で指定できます。

次に示すものに関しては、引用符で囲んだファイル名およびパスを使用できません。

- SAS カタログ
- MDDB 参照および FDB 参照
- COPY プロシジャの SELECT ステートメントや DATASETS プロシジャのステートメントなど、ライブラリ参照名を入力しない場合
- PROC SQL
- ストアド DATA ステッププログラムまたは SAS ビュー
- スクリーンコントロール言語(SCL)の OPEN 関数

ライブラリ参照名を使用せずに SAS データファイルにアクセスする DATA ステートメントの例を次の表に示します。

表 24.2 *ライブラリ参照名を使用せずに SAS データファイルにアクセスする DATA ステートメントの例*

動作環境	例
DOS、Windows	`data 'c:\root\mystuff\sasstuff\work\myfile';`
UNIX	`data '/u/root/mystuff/sastuff/work/myfile';`

動作環境	例

z/OS	`data 'user489.mystuff.saslib(member1)';`
	`  /* bound SAS library */`
	
	`data '/mystuff/sasstuff/work/myfile';`
	`  /* UNIX file system library */`

動作環境コマンド

動作環境コマンドを使用すると、対応する動作環境のファイル、または SAS ライブラリを構成する SAS ファイルに対して、コピー、名前の変更、削除を実行できます。ただし、ファイルの一貫性を維持するには、SAS ライブラリが動作環境に実装される仕組みを理解している必要があります。たとえば、一部の動作環境では、SAS データセットおよびそれに割り当てられているインデックスに対して、別々のファイルとして、コピー、削除、名前の変更を実行できる場合があります。SAS データセット名を変更して、インデックスの名前を変更しない場合は、データセットは破損したものと見なされます。

注意:

動作環境のコマンドを使用することによって、ファイルが破損する場合があります。 SAS ファイルを管理する場合は、常に SAS Utilities を使用してください。

25 章

SAS データセット

SAS データセットの定義

SAS データセットとは、SAS System が作成し処理できる SAS ライブラリ内に保存されている SAS ファイルのことです。SAS データセットには、SAS System が処理できる、オブザベーション(行)と変数(列)からなるテーブル形式で構成されたデータ値が含まれています。SAS データセットには、変数のデータ型や長さ、データの作成に使用されたエンジンなどに関するディスクリプタ情報も含まれています。

SAS デートセットは次のいずれかになります。

SAS データファイル
: データとディスクリプタ情報の両方を含んでいます。SAS データファイルはメンバタイプ DATA を持つ SAS ファイルとして管理されます。詳細については、26 章, “SAS データファイル” (597 ページ)を参照してください。

SAS ビュー
他のソースに含まれているデータを指す仮想データセットです。SAS ビューは、メンバタイプ VIEW を持つ SAS ファイルとして管理されます。詳細については、27 章, “SAS ビュー” (665 ページ)を参照してください。

注: SAS データセットという用語は、SAS ビューと SAS データファイルを同じ方法で使用できる場合に使用します。

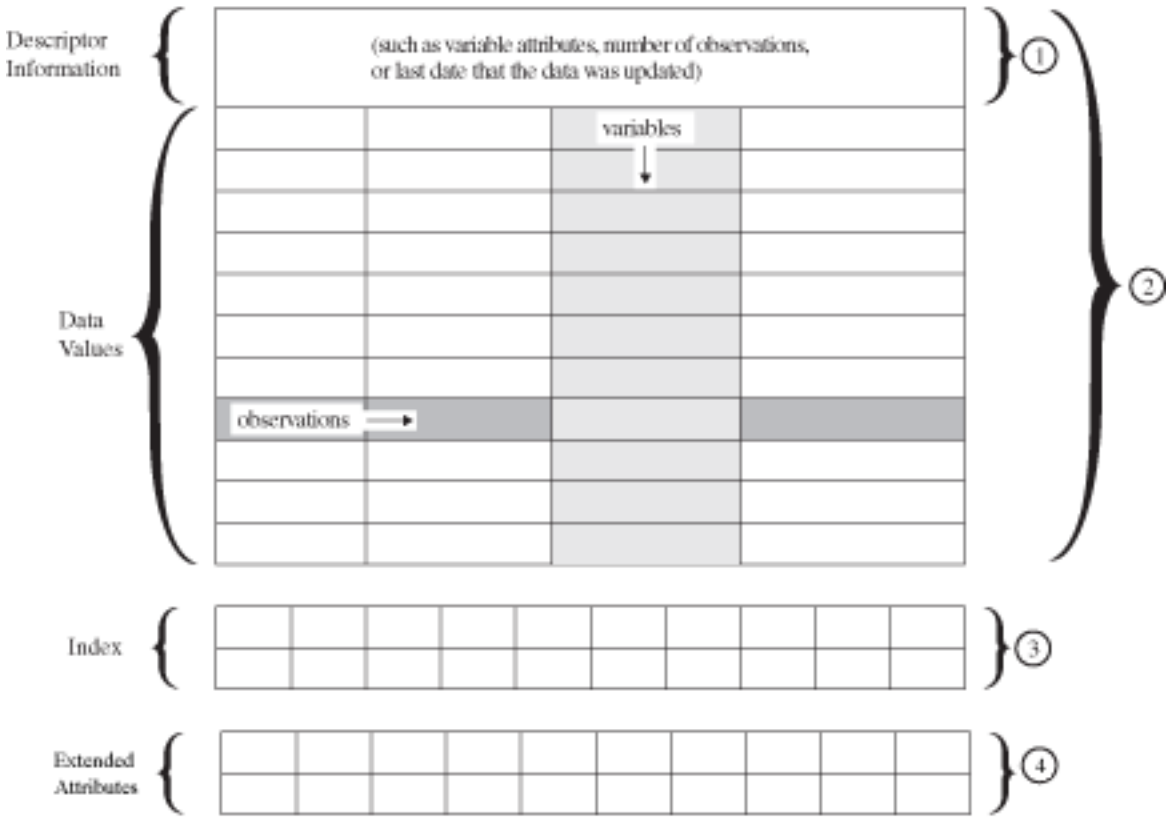
SAS データセットのディスクリプタ情報

SAS データセットのディスクリプタ情報は、ファイルの自己ドキュメント化を行います。すなわち、各 SAS データセットは、自分自身に関する属性と、それが含む変数に関する属性を提供します。データが SAS データセット形式で提供される場合、プログラムステートメントで、同データセットの属性や変数の属性を指定する必要がありません。それらの情報は、データセットから直接取得します。

ディスクリプタ情報には、オブザベーションの数、オブザベーションの長さ、データセットが最後に更新された日付など、SAS データセットについての属性情報が格納されています。また、ディスクリプタ情報には、変数に関する変数名、変数の種類、長さ、フォーマット、ラベル、インデックスの有無などの属性も格納されます。

次の図は、SAS データセットに含まれている論理コンポーネントを示しています。

図 25.1 SAS データセットの論理コンポーネント



次に示す各項目の説明は、上図の番号と対応しています。

1. SAS ビュー(メンバタイプ VIEW)は、他のデータセットや外部ファイルに格納されているディスクリプタ情報およびデータ値を参照します。
2. SAS データファイル(メンバタイプ DATA)は、ディスクリプタ情報とデータ部を格納します。SAS データセットのメンバタイプは、DATA (SAS データファイル)または VIEW (SAS ビュー)のどちらかです。
3. インデックスとは、ユーザーが SAS データファイル用に作成する独立したファイルです。インデックスを使用すると、特定のオブザベーションに直接アクセスできます。インデックスファイルの名前は、そのデータファイルの名前と同じであり、メンバタイプは INDEX になります。インデックスを使用すると、特定のオブザベーションへのアクセスが高速になります。大きなデータセットに対しては特に有効です。
4. 拡張属性は、データセットまたは変数(列)に関して定義されるメタデータです。拡張属性は、DATASETS プロシジャを使用して作成され、名前と値のペアで表現されます。

データセット名

データセット名の使用場所

SAS データセットを DATA ステップまたは PROC ステップの入力として使用するには、次のステートメントやオプションでデータセット名を指定します。

- SET ステートメント
- MERGE ステートメント
- UPDATE ステートメント
- MODIFY ステートメント
- SAS プロシジャの DATA=オプション
- OPEN 関数

SAS データセット名の割り当て方法とタイミング

SAS データセットを作成する場合、データセット名を割り当てます。DATA ステップで作成する出力データセットは、DATA ステートメントで割り当てます。PROC ステップで作成する出力データセットは、プロシジャステートメントまたは OUTPUT ステートメントで割り当てます。出力データセット名を指定しない場合は、SAS System によってデフォルト名が割り当てられます。

SAS ビューを作成する場合、次のいずれかを使用してデータセット名を割り当てます。

- SQL プロシジャ
- ACCESS プロシジャ
- DATA ステートメントの VIEW=オプション

注: 1 つのプログラムステートメントの中で、SAS データファイルと SAS ビューの両方をデータセットとして指定することができます。ただし、メンバタイプは指定できないため、SAS System では、どちらを処理するかをプログラムステートメントから特定することはできません。したがって、同一のライブラリにある SAS ビューと SAS データファイルに対して同一の名前を割り当てることはできません。

データセット名の構成要素

SAS データセット名の完全な形式は、3 つの要素から構成されます。ユーザーが最初の 2 つの要素のみを割り当てると、SAS System が 3 つ目の要素を自動的に割り当てます。SAS データセット名の完全な形式は、次に示す 3 レベル名です。

```
libref.SAS-data-set.membertype
```

SAS データセット名を構成する各要素は次のとおりです。

libref
SAS ライブラリの物理的な格納場所に関連付けられている論理名です。

SAS-data-set
データセット名を指定します。データセット名の最大長は、バージョン 7 以降の Base SAS Engine では 32 バイトです。それ以前の SAS バージョンでは、8 バイトです。

membertype
SAS System によって割り当てられるメンバタイプです。メンバタイプは、SAS データファイルの場合は DATA、SAS ビューの場合は VIEW になります。

プログラムステートメントの中で SAS データセットを参照するときは、1 レベル名または 2 レベル名を使用します。1 レベル名は、データセットが Work などの一時ライブラリにある場合に使用します。また、予約ライブラリ参照名 User が割り当てられている場合、データセットが永久ライブラリ User 内に存在するならば、1 レベル名を使用できます。2 レベル名は、データセットがユーザーの作成した別の永久ライブラリにある場合に使用します。2 レベル名は、ライブラリ参照名とデータセット名で構成されます。1 レベル名は、データセット名だけで構成されます。

2 レベルの SAS データセット名

永久 SAS ライブラリ内に保存される SAS データセットに対して作成、読み込み、書き出しを行う場合に最もよく使用される形式は、次に示す 2 レベル名です。

libref.SAS-data-set

新しい SAS データセットを作成する場合、ライブラリ参照名はデータセットの格納先を示します。既存のデータセットを参照する場合は、ライブラリ参照名によってデータセットの検索場所を SAS System に指示します。次の例は、SAS ステートメントの中で 2 レベル名を使用する方法を示しています。

```
data revenue.sales;

proc sort data=revenue.sales;
```

1 レベルの SAS データセット名

ライブラリ参照名を省略し、次の形式で 1 レベル名を使用してデータセットを参照することができます。

SAS-data-set

1 レベル名のデータセットは、Work と User という 2 つの特殊な SAS ライブラリのどちらかに自動的に割り当てられます。このデータセットは、通常は一時ライブラリ Work に割り当てられ、SAS ジョブまたは SAS セッションの終了時に削除されます。SAS ライブラリとライブラリ参照名 User を関連付けた場合や、USER=システムオプションを使用して User ライブラリを設定した場合は、1 レベル名のデータセットはそのライブラリに格納されます。User および Work ライブラリの詳細については、24 章, “SAS ライブラリ” (565 ページ) を参照してください。次の例は、SAS ステートメントの中で 1 レベル名を使用する方法を示しています。

```
/* create perm data set in location of USER=option*/
options user='c:\temp'
data test3;

/* create perm data set in current directory */
data 'test3';

/* create a temp data set in WORK directory if USER= is not specified*/
data stratifiedsample1;
```

データセットリスト

DATASETS プロシジャと DATA ステップの MERGE および SET ステートメントで、データセットのリストを使うと、既存のデータセットのグループを簡単に参照できます。このデータセットのリストは、番号付き範囲リストか、またはコロン(名前の接頭辞)リストとして指定します。

- 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 2 つのリストは同じデータセットを参照します。

```
sales1 sales2 sales3 sales4

sales1-sales4
```

  注: 最初のデータセット名の数値接尾辞がゼロで始まる場合、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数値接尾辞の桁数に等しいかまたはそれよりも大きくなければなりません。それ以外の場合、エラーが発生します。たとえば、データセットリスト *sales001–sales99* や *sales01–sales9* を指定するとエラーが発生します。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の数値接尾辞がゼロで始まらない場合は、最初と最後の各データセット名の数値接尾辞の桁数が等しくなくてもかまいません。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。

- コロン(名前の接頭辞)リストでは、同じ文字(または文字列)で始まる名前を持つ一連のデータセットを指定します。たとえば、次の 2 つのリストは同じデータセットを参照します。

```
abc:

abc1 abc2 abcr abcx
```

DATASETS プロシジャ内では、データセットリストを次のステートメントで使用できます。

- COPY SELECT ステートメント
- COPY EXCLUDE ステートメント
- DELETE ステートメント
- REPAIR ステートメント
- REBUILD ステートメント
- MODIFY SORTEDBY ステートメント内で指定する変数

DATA ステップ内でのデータセットリストを使用する方法の詳細については、“MERGE Statement” (*SAS Statements: Reference*)と“SET Statement” (*SAS Statements: Reference*)を参照してください。

特殊な SAS データセット

ヌルデータセット

SAS データセットを作成せずに、DATA ステップを実行する場合、データセット名にキーワード_NULL_を指定します。次のステートメントは、出力データセットを作成せずに DATA ステップを開始します。

```
data _null_;
```

データセット名にキーワード_NULL_を指定すると、DATA ステップは新しいデータセットを作成する場合と同じように実行されます。しかし、オブザベーションや変数は出力データセットに書き出されません。レポート作成などを目的として DATA ステップを使用する場合は、_NULL_を使用することで、コンピュータのリソースを効率的に利用できます。レポート作成などで、DATA ステップの出力を SAS データセットとして格納する必要がない場合に使用できます。

デフォルトデータセット

_LAST_という予約名を使用すると、最後に作成された SAS データセットを参照できます。入力データセットを指定しないで DATA ステップまたは PROC ステップを実行すると、デフォルトではデータセット_LAST_が使用されます。データセット_LAST_は、一部の関数でも使用されます。

LAST=システムオプションを使用すると、キーワード_LAST_が参照するデータセットを指定できます。指定したデータセットは、新しいデータセットを作成するまでデフォルトデータセットとして使用されます。同一のデータセットを繰り返し使用する PROC ステップを多く含むプログラムで既存の永久データセットを使用する場合、_LAST_=システムオプションを使用して、デフォルトデータセットを指定すると便利です。_LAST_=システムオプションを使用すると、各プロシジャステートメント内で SAS データセット名を指定しなくてもすみます。次の OPTIONS ステートメントでは、デフォルト SAS データセットを指定しています。

```
options _last_=schedule.january;
```

自動命名規則

DATA ステートメントの中で、SAS データセット名も予約名_NULL_も指定しなかった場合は、DATA1、DATA2 などの名前を持つデータセットが Work ライブラリまたは User ライブラリに自動的に作成されます。この機能のことを、DATA*n* 命名規則と呼びます。次のステートメントを実行すると、DATA*n* 命名規則を使用した SAS データセットが作成されます。

```
data;
```

並べ替えられたデータセット

ソートインジケータ

ソートインジケータについて

データセットを並べ替えると、そのデータセットのディスクリプタ情報にソートインジケータが追加されます。SORTEDBY=データセットオプションを使用すると、データセットの永久的な並べ替えが行われずに、ソートインジケータが更新されます。SORTEDBY=データセットオプションを使用すると、**Sortedby** および **Validated** 並べ替え情報が更新されます。

ソートインジケータには、SAS データセットに関する次の並べ替え情報の一部または全部が含まれています。

- データセットがどの変数を基準として、どのように並べ替えられているか
- 変数の並べ替え順序(昇順か降順か)
- 文字変数に使用される文字セット
- 文字データの順序に使用される照合順序
- SORTSEQ=LINGUISTIC オプションを使ってデータセットを並べ替える場合の照合規則
- BY グループにオブザベーションが 1 つだけ存在するかどうか(NODUPKEY オプションを使用)
- 重複したオブザベーションが隣接して存在しないかどうか(NODUPKEY オプションを使用)
- データセットが検証済みであるかどうか

ソートインジケータは、SORT プロシジャ、SQL プロシジャの ORDER BY 句、DATASETS プロシジャの MODIFY ステートメント、SORTEDBY=データセットオプションのいずれかを使用してデータセットを並べ替えた場合に設定されます。SORT プロシジャまたは SQL プロシジャを使用してデータセットを並べ替えた場合、これは SAS System による並べ替えであるため、CONTENTS プロシジャ出力では **Validated** 並べ替え情報が YES であると表示されます。SORTEDBY=データセットオプションを使用してデータセットを並べ替えた場合、これはユーザーによる並べ替えであるため、CONTENTS プロシジャ出力では、**Validated** 並べ替え情報は NO に設定され、**Sortedby** 並べ替え情報は SORTEDBY=データセットオプションに指定された変数の値で更新されます。

データセットは SAS System の外部でも並べ替えできます。その場合、SORTEDBY=データセットオプションまたは DATASETS プロシジャの MODIFY ステートメントを使って、並べ替え順序をソートインジケータに追加することができます。この場合、そのデータセットは検証済みになりません。詳細については、“データセットが並べ替え済みであることの検証” (593 ページ)を参照してください。

ソートインジケータ情報を表示するには、CONTENTS プロシジャか、または DATASETS プロシジャの CONTENTS ステートメントを使用します。CONTENTS プロシジャを使用してソートインジケータ情報を表示する例を次に示します。

例 1: 並べ替えを行わない場合

この最初の例では、並べ替えを行わずにデータセットを作成しています。

```
options yearcutoff=1920;
libname myfiles 'C:\My Documents';

data myfiles.sorttest1;
   input priority 1. +1 indate date7.
         +1 office $ code $;
   format indate date7.;
   datalines;
1 03may11 CH J8U
1 21mar11 LA M91
1 01dec11 FW L6R
1 27feb10 FW Q2A
2 15jan11 FW I9U
2 09jul11 CH P3Q
3 08apr10 CH H5T
3 31jan10 FW D2W
;
proc contents data=myfiles.sorttest1;
run;
```

CONTENTS プロシジャ出力には、並べ替えが行われていないことが示されます。これは、SAS System による並べ替えが行われておらず、ユーザーもデータの並べ替えを指定していないためです。

アウトプット 25.1 *Sorttest1 データセットの内容 – 並べ替えを行わない場合*

The SAS System

The CONTENTS Procedure

Data Set Name	MYFILES.SORTTEST1	Observations	8
Member Type	DATA	Variables	4
Engine	V9	Indexes	0
Created	04/10/2014 12:50:17	Observation Length	32
Last Modified	04/10/2014 12:50:17	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

例 2:SORTEDBY=データセットオプションを使用した場合

この 2 番目の例では、DATA ステートメントで SORTEDBY=データセットオプションを使用して、データセットを作成しています。

```
options yearcutoff=1920;
libname myfiles 'C:\My Documents';
```

```
data myfiles.sorttest1 (sortedby=priority descending indate);
   input priority 1. +1 indate date7.
         +1 office $ code $;
   format indate date7.;
   datalines;
1 03may01 CH J8U
1 21mar01 LA M91
1 01dec00 FW L6R
1 27feb99 FW Q2A
2 15jan00 FW I9U
2 09jul99 CH P3Q
3 08apr99 CH H5T
3 31jan99 FW D2W
;
proc contents data=myfiles.sorttest1;
run;
```

CONTENTS プロシジャの出力にはデータセットが並べ替え済みであることが示されていることに注意してください。このため、ソートインジケータ情報を含む **Sort Information** セクションが作成されます。**Sort Information** セクション内の **Sortedby** 情報には、このデータセットが PRIORITY 変数を使用して並べ替えられた後、INDATE 変数を使用して降順に並べ替えられていることが示されています。このデータセットは SORTEDBY=データセットオプションを使用して並べ替えられているため、**Validated** 情報は NO に設定されます。同データセットの **Character Set** 情報は ANSI に設定されています。

アウトプット 25.2 *Sorttest1 データセットの内容 – 並べ替え済みの場合*

The SAS System

The CONTENTS Procedure

Data Set Name	MYFILES.SORTTEST1	Observations	8
Member Type	DATA	Variables	4
Engine	V9	Indexes	0
Created	04/10/2014 12:51:54	Observation Length	32
Last Modified	04/10/2014 12:51:54	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

アウトプット 25.3 *Sorttest1 データセットの内容 – 並べ替え済みの場合*

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
4	code	Char	8	
2	indate	Num	8	DATE7.
3	office	Char	8	
1	priority	Num	8	

Sort Information	
Sortedby	priority DESCENDING indate
Validated	NO
Character Set	ANSI

例 3:SORT プロシジャを使用する場合

この 3 番目の例では、SORT プロシジャを使用してデータセットを並べ替えています。

```
options yearcutoff=1920;
libname myfiles 'C:\My Documents';

data myfiles.sorttest1;
   input priority 1. +1 indate date7.
         +1 office $ code $;
   format indate date7.;
   datalines;
1 03may01 CH J8U
1 21mar01 LA M91
1 01dec00 FW L6R
1 27feb99 FW Q2A
2 15jan00 FW I9U
2 09jul99 CH P3Q
3 08apr99 CH H5T
3 31jan99 FW D2W
;
proc sort data=myfiles.sorttest1;
     by priority descending
indate;
run;

proc contents data=myfiles.sorttest1;
run;
```

CONTENTS プロシジャの出力にはデータセットが並べ替え済みであることが示されていることに注意してください。このため、ソートインジケータ情報を含む **Sort Information** セクションが作成されます。**Sort Information** セクション内の **Sortedby** 情報には、このデータセットが PRIORITY 変数を使用して並べ替えられた後、INDATE 変数を使用して降順に並べ替えられていることが示されています。このデー

タセットは SORT プロシジャを使って並べ替えられているため、**Validated** 情報は YES に設定されています。同データセットの **Character Set** 情報は ANSI に設定されています。

アウトプット 25.4 *Sorttest1 データセットの内容 – 検証済み並べ替えの場合*

The SAS System

The CONTENTS Procedure

Data Set Name	MYFILES.SORTTEST1	Observations	8
Member Type	DATA	Variables	4
Engine	V9	Indexes	0
Created	04/10/2014 12:54:33	Observation Length	32
Last Modified	04/10/2014 12:54:33	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

アウトプット 25.5 *Sorttest1 データセットの内容 – 検証済み並べ替えの場合*

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
4	code	Char	8	
2	indate	Num	8	DATE7.
3	office	Char	8	
1	priority	Num	8	

Sort Information

Sortedby	priority DESCENDING indate
Validated	YES
Character Set	ANSI

SAS でのソートインジケータを使用したパフォーマンスの向上について

ソートインジケータにより提供される並べ替え情報は、パフォーマンス向上のために内部的に使用されます。ソートインジケータを使用してパフォーマンスを向上させるにはいくつかの方法があります。

- SAS System はソートインジケータを使用して、過去に並べ替えが行われたかどうかを検証します。SORT プロシジャまたは SQL プロシジャの ORDER BY 句を使用して過去に並べ替えが実施されている場合、SAS System はもう一度並べ替えを実行しません。
  - SORT プロシジャは、並べ替えが行われると、ソートインジケータを設定します。SORT プロシジャは、データセットの並べ替えを行う前にソートインジケータをチェックすることにより、データの不要な再度の並べ替えを防ぎます。詳細については、*Base SAS Procedures Guide* にある SORT プロシジャの説明を参照してください。
  - SQL プロシジャは、ソートインジケータを使用することにより、クエリをより効率的に処理し、結合を行う前に内部ソートが必要であるかどうかを判定します。詳細については、*Base SAS Procedures Guide* にある SQL プロシジャの説明を参照してください。
- インデックスの作成時にソートインジケータを使用すると、SAS System は、当該データファイル内のソートインジケータをチェックすることにより、そのデータがキー変数を使用して昇順で並べ替え済みであるかどうかを判定します。ソートインジケータの値が昇順になっている場合、SAS System はインデックスファイルの値を並べ替えません。詳細については、“SAS インデックスについて” (632 ページ)を参照してください。
- インデックスの指定なしで WHERE 式を処理する場合、SAS System はまずソートインジケータをチェックします。**Validated** 並べ替え情報が YES である場合、SAS System は、その WHERE 式を満足する値が存在しなくなった時点でファイルの読み取りを停止します。
- WHERE 式でインデックスが選択されている場合、同データセットのソートインジケータは、インデックスに指定されている順序を反映するように変更されます。
- BY グループ処理で、データセットが BY 変数で並べ替え済みである場合、その BY 変数でデータセットがインデックス付けされている場合であっても、SAS System はインデックスを使用しません。
- **Validated** 並べ替え情報が YES に設定されている場合、SAS System はもう一度並べ替えを行う必要がありません。

データセットが並べ替え済みであることの検証

処理の一環として、データが並べ替え済みであることを要求する SAS プロシジャは、ソートインジケータ情報をチェックします。ソートインジケータは、SORT プロシジャ、SQL プロシジャの ORDER BY 句、DATASETS プロシジャの MODIFY ステートメント、SORTEDBY=データセットオプションのいずれかを使用してデータセットを並べ替えた場合に設定されます。SORT プロシジャまたは SQL プロシジャを使用してデータセットを並べ替えた場合、CONTENTS プロシジャ出力では **Validated** 並べ替え情報が YES であると表示されます。SORTEDBY=データセットオプションを使用してデータセットを並べ替えた場合、CONTENTS プロシジャ出力では **Validated** 並べ替え情報が NO であると表示されます。CONTENTS プロシジャ出力の例については、“例 1: 並べ替えを行わない場合” (588 ページ)、“例 2:SORTEDBY=データセットオプションを使

用した場合” (589 ページ)および“例 3:SORT プロシジャを使用する場合” (591 ページ)を参照してください。

SORTVALIDATE システムオプションを使用すると、データセットのソートインジケータがユーザーによるデータセットの並べ替えの指定を示している場合に、SORT プロシジャでデータセットが正しく並べ替えられていることを検証するかどうかを指定できます。ユーザーが並べ替え順序を指定するには、DATA ステートメントで SORTEDBY=データセットオプションを使用するか、または DATASETS プロシジャの MODIFY ステートメントで SORTEDBY=オプションを使用します。ユーザーがソートインジケータを設定した場合、SAS System はデータセットが BY ステートメント内の変数に従って並べ替えられているかどうかを完全には判定できなくなります。

SORTVALIDATE システムオプションを設定した場合、データセットのソートインジケータがユーザーにより設定されているならば、SORT プロシジャは、各オブザベーションのシーケンスチェックを実施することにより、同データセットが BY ステートメント内の変数に従って並べ替えられていることを確認します。データセットが正しく並べ替えられていない場合、SAS System はそのデータセットを並べ替えます。

シーケンスチェックの終了時または並べ替えの終了時に、SORT プロシジャは、ソートインジケータの **Validated** 並べ替え情報を YES に設定します。並べ替えを実行する場合、SORT プロシジャは、ソートインジケータの **Sortedby** 並べ替え情報を、BY ステートメント内の変数へと更新します。出力データセットを指定した場合、その出力データセット内のソートインジケータの **Validated** 並べ替え情報が YES に設定されます。並べ替えが必要ない場合、データセットが出力データセットにそのままコピーされます。データセットの検証に関する詳細については、“SORTVALIDATE System Option” (*SAS System Options: Reference*)を参照してください。

データセット管理のツール

SAS データセットに関する情報を、コピー、名前変更、削除、取得するには、SAS ライブラリの場合と同様にウィンドウ、プロシジャ、関数、オプションを使用します。これらのウィンドウやプロシジャの一覧については、24 章, “SAS ライブラリ” (565 ページ)を参照してください。

また、SAS データセットの操作に利用できる関数も提供されています。詳細については、各関数の説明を参照してください。

SAS データセットの表示と編集

VIEWTABLE ウィンドウでは、データセットの表示、編集、作成を行うことができます。このウィンドウには、次の 2 つの表示モードがあります。

テーブルビュー
: 表形式で、データセット内の複数のオブザベーションを表示します。

フォームビュー
: レイアウトフォームを使用して、データを一度に 1 オブザベーションずつ表示します。

データセットの表示は、カスタマイズすることができます。たとえば、データを並べ替える、列の色とフォントを変更する、変数名の代わりに変数ラベルを表示する、変数を削除または追加する、などの操作が行えます。また、既存の DATAFORM カタログエントリを呼び出して、定義済み変数、定義済みデータセット、定義済みビューア属性を適用することもできます。

データセットを表示するには、**ツール** ⇨ **テーブルエディタ**を選択します。これにより、**VIEWTABLE** または **FSVIEW** (z/OS)ウィンドウが表示されます。**エクスプローラ**ウィンドウでデータセットをダブルクリックしても同じ画面を表示できます。

VIEWTABLE ウィンドウでサポートされる SAS ファイルは次のとおりです。

- SAS データファイル
- SAS ビュー
- MDDB ファイル

詳細については、Base SAS の **VIEWTABLE** ウィンドウのオンラインヘルプを参照してください。

26 章

SAS データファイル

SAS データファイルの定義

SAS データファイルとは、SAS データセットの一種であり、データ値とディスクリプタ情報の両者を格納しているものです。SAS データファイルは、メンバタイプ DATA を持つ SAS ファイルとして管理されます。SAS データファイルには次の 2 種類があります。

ネイティブ SAS データファイル
: SAS System によって作成されたデータ値とディスクリプタ情報を格納した形式のファイルです。

インターフェイス SAS データファイル
: SAS System 以外のソフトウェアによって作成されたデータ格納ファイルです。SAS System は、Oracle、DB2、SYBASE、ODBC、BMDP、SPSS、OSIRIS などのソフトウェアによって作成されたファイルのデータを SAS System で読み書きするためのエンジンを搭載しています。これらのファイルがインターフェイスデータファイルで

す。エンジンを経由してこれらのデータにアクセスする場合、SAS System は、インターフェイスデータファイルを SAS データセットとして認識します。

注: どのような種類のインターフェイスデータファイルにアクセスできるエンジンを利用できるかは、サイトのライセンス契約によって異なります。利用可能なエンジンを確認するには、システム管理者に問い合わせてください。SAS System のマルチエンジンアーキテクチャ(MEA)の詳細については、35 章, “SAS エンジン” (737 ページ)を参照してください。

SAS データファイルと SAS ビューの違い

*SAS データファイル*と *SAS ビュー*という用語は、多くの場合同じ意味で使用されますが、両者は厳密には異なっています。

最大の違いは、値が格納される場所です。
SAS データファイルは、ディスクリプタ情報とデータ部を物理的に格納する SAS データセットです。一方、SAS ビューは、別のファイルに格納されているデータ部およびディスクリプタ情報の取り出しに必要な情報だけを格納する SAS データセットです。SAS System によりデータが取り出されると、それ以降は、同データを DATA ステップで処理できるようになります。

SAS データファイルは静的です。これに対して、SAS ビューは動的です。
SAS データファイルを後続の PROC ステップで参照するときは、データファイルが作成された時点または最後に更新された時点のデータ値が表示されます。SAS ビューを PROC ステップで参照すると、そこでビューが実行され、ビューが定義された時点ではなく現時点のデータ値が表示されます。

SAS データファイルは、テープまたはその他の記憶媒体上に作成することが可能です。
SAS ビューはテープ上に保存できません。SAS ビューは、その動的な特性のため、ディスクドライブなどのランダムアクセス方式の記憶デバイス上にあるデータファイルからデータを取得する必要があります。SAS ビューは、テープドライブなどのシーケンシャルアクセス方式の記憶デバイス上に格納されているファイルからデータを取得することはできません。

SAS ビューは読み取り専用です。
SAS ビューに書き込みを行うことはできませんが、一部の SQL ビューは更新することができます。

SAS データファイルは監査証跡を持つことが可能です。
監査証跡とは、SAS データファイルに対する変更を記録するオプションの SAS ファイルです。オブザベーションが追加、削除、更新されるたびに、修正者、修正内容、修正日時に関する履歴情報が監査証跡ファイルに記録されます。

SAS データファイルは世代を持つことが可能です。
世代とは、特定の SAS データファイルの複数のコピーを保持する機能です。これらの複数のコピーは、同じデータファイルの異なるバージョンを表しており、バージョンが置き換えられるたびに旧バージョンのアーカイブとして作成されるものです。

SAS データファイルは、一貫性制約を持つことが可能です。
SAS データファイルを更新する場合、一貫性制約を使用して、データを特定の基準に合わせることができます。SAS ビューの場合、ビューが参照するデータファイルに一貫性制約を割り当てることによってのみ、間接的にデータを基準に合わせることができます。

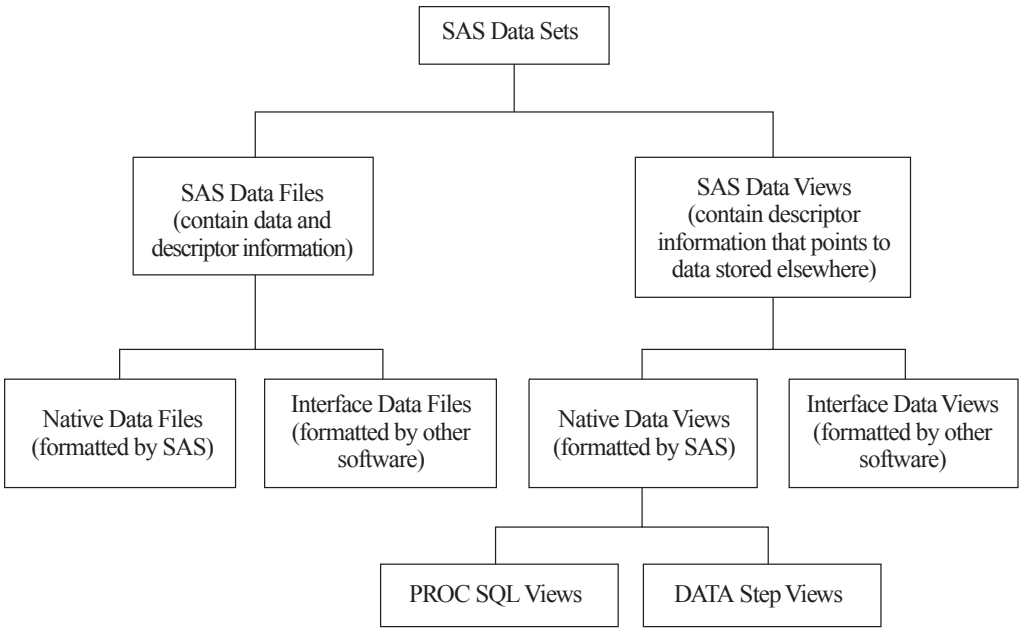
SAS データファイルには、インデックスを付けることが可能です。
インデックスを付けると、SAS データファイルのデータをより高速に検索できます。SAS ビューには、インデックスを付けることはできません。

SAS データファイルは、暗号化が可能です。
暗号化すると、物理ファイルにセキュリティ層が追加されます。SAS ビューは暗号化できません。

SAS データファイルは、圧縮が可能です。
圧縮すると、物理ファイルを格納するディスク領域を減らすことができます。SAS ビューは圧縮できません。

次の図に、SAS データファイル(ネイティブデータファイルとインターフェイスデータファイル)、SAS ビュー(ネイティブビューとインターフェイスビュー)、および両者の関係を示します。

図 26.1 *SAS データセットの種類*



SAS データファイルのオブザベーションカウントについて

オブザベーションカウントの定義

SAS データファイル内のオブザベーションカウントとは、同ファイル内に現在あるオブザベーション(行)の数と、削除されたオブザベーションの数を合計した総数のことです。オブザベーションカウントは、特定の SAS データファイルに対して CONTENTS プロシジャまたは DATASETS プロシジャの CONTENTS ステートメントを実行することで一覧表示できるファイル属性のうちの一つです。プロシジャ出力では、オブザベーションカウントは、**オブザベーション数**と**削除されたオブザベーション数**の各フィールドに表示される値の合計になります。オブザベーションカウントを知ることは、ファイルサイズの管理や、必要なディスクスペースを推定する場合に役立ちます。また、SAS データファイルにはカウント可能な最大オブザベーション数が存在します。これは、動作環境の long 型整数のサイズにより決定されます。

最大オブザベーションカウント

SAS データセットでカウントできる最大オブザベーション数は、動作環境における long 型整数データのサイズにより決定されます。

- 32 ビット長整数を使用する動作環境では、最大数は 2^{31}–1、つまり約 20 億オブザベーション(2,147,483,647)です。
- 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、つまり約 920 京になります。

このため、64 ビット長整数を使用する動作環境では、最大オブザベーションカウントに到達する可能性は非常に低くなります。一方、32 ビット長整数を使用する動作環境では、約 20 億の最大オブザベーションカウントに到達することは珍しくありません。

オブザベーションカウントを 32 ビット長整数で格納するような内部データ表現を持つ SAS 9.4 の動作環境としては、次のプラットフォームが挙げられます。

- 32 ビットプラットフォーム上の Microsoft Windows
- 64 ビットエディションの Microsoft Windows この 64 ビット対応動作環境では、32 ビットアプリケーションとの互換性を保持するために、long 型整数データとしては 32 ビットモデルが使用されます。
- 32 ビットプラットフォーム上の z/OS

最大オブザベーションカウントに達した場合の SAS System の処理

SAS データファイルが最大オブザベーションカウントに達した場合、SAS System の処理を継続できるかどうかは、同ファイルがインデックスを持つかどうか、またはインデックスを使用する一貫性制約を持つかどうかにより決まります。

- SAS データファイルがインデックスを持つか、またはインデックス(一意のキー、主キー、外部キー)を使用する一貫性制約を持つ場合、操作が最大オブザベーションカウントに達すると、エラーメッセージが発行されます。次に例を示します。

  ```
  ERROR:File MYFILES.BIGFILE contains 2G -1 observations and cannot hold more because it c
  ```

  SAS バージョン 9 では、最大オブザベーションカウントを超えて操作が行われようとした場合でも、SAS データファイルは破損することはありません。ただし、ファイルの処理を続行するには、明示的な措置を実施する必要があります。

- SAS データファイルがインデックスを持たず、インデックスを使用する一貫性制約も持たない場合、シーケンシャルな処理は続行され、追加のオブザベーションが受け入れられます。ただし、同データファイルでは、それらの追加のオブザベーションカウントを格納できず、追加のオブザベーション番号も保持しません。このため、オブザベーション番号を必要とする操作は使用できません。ファイルが最大オブザベーションカウントに達したか、またはそれを超えたことを示すようなメッセージは発行されません。

  インデックスを持たず、インデックスを使用する一貫性制約も持たない SAS データファイルが最大オブザベーションカウントを超えた場合、制限を受ける操作や機能が存在します。それらの操作や機能の一部を次に示します。制限を受ける操作や機能の完全な一覧については、SAS テクニカルサポートにお問い合わせください。

  - オブザベーションカウントを返す SAS プロシジャ(PRINT プロシジャや CONTENTS プロシジャなど)は、オブザベーションの数として欠損値を返します。欠損値はピリオド(.)で表されます。

- オブザベーションカウントに依存する SAS プロシジャ(SORT プロシジャや COMPARE プロシジャ)は、予測できない結果を返します。
- オブザベーションカウントを更新する操作はサブミットできません。オブザベーションを削除することでは、オブザベーションカウントのリセットは行えません。
- オブザベーションカウントが保持されなくなったファイルの圧縮を要求した場合、圧縮率は計算できません。
- インデックスや一貫性制約を作成できません。
- 動作環境間で CEDA 処理を実施した場合の動作を次に示します。SAS 9.3 以降では、動作環境間での CEDA 処理が改善されていることに注意してください。

表 26.1 *動作環境間での CEDA 処理*

動作環境	実行する操作	SAS 9.3 以前のバージョンでの動作	現在の動作
32 ビット長整数の動作環境	32 ビット長で表現できる最大値を超える数のオブザベーションを含む 64 ビット長整数対応ファイルのオープン	オープンに失敗します。	32 ビットカウンタが改善されているため、同ファイルをオープンできます。
32 ビット長整数の動作環境	32 ビット長で表現できる最大値を超える数のオブザベーションを含む 64 ビット長整数対応ファイルの作成	出力処理が停止します。	32 ビットカウンタが改善されているため、同ファイルを作成できます。
64 ビット長整数の動作環境	32 ビット長で表現できる最大値を超える数のオブザベーションを含む 32 ビット長整数対応ファイルのオープン	オープンに失敗します。	ファイルはオープンできますが、オブザベーション番号が使用できないため機能は限定されます。
64 ビット長整数の動作環境	32 ビット長で表現できる最大値を超える数のオブザベーションを含む 32 ビット長整数対応ファイルの作成	出力処理が停止します。ファイルは作成されません。	32 ビット長で表現できる最大値と同じ数のオブザベーションを含むファイルが作成されます。

最大オブザベーションカウントを超過したデータセットの回復

インデックスを持つ、またはインデックスを使用する一貫性制約を持つ SAS データファイルが、カウント可能なオブザベーションの最大数に達したかまたはそれを超えた場合、処理を続行するためには明示的な措置を実施する必要があります。

- この場合、インデックスまたは一貫性制約を削除することにより、処理を続行できます。ただし、最大オブザベーションカウントを超えているため、機能は限定されます。インデックスまたは一貫性制約を削除するには、DATASETS プロシジャか SQL プロシジャを使用します。*Base SAS Procedures Guide* を参照してください。

- インデックスや一貫性制約を保持するためには、SAS データファイルを再作成する必要があります。SAS データファイルを作成する場合、SAS 9.4 からは、デフォルトで拡張オブザベーションカウントの付いたファイルが作成されます。“32 ビット SAS データファイルのオブザベーションカウントの拡張” (658 ページ)を参照してください。

インデックスを持たない、またはインデックスを使用する一貫性制約を持たない SAS データファイルでは、カウント可能なオブザベーションの最大数に達したかまたはそれを超えた場合でも、そのファイルが最大オブザベーションカウントに達したか、またはそれを超えたことを示すメッセージは出されません。ただし、この場合、同ファイルの機能は制限されます。拡張オブザベーションカウントが付いた SAS データファイルを再作成して、機能を回復できます。“32 ビット SAS データファイルのオブザベーションカウントの拡張” (658 ページ)を参照してください。

監査証跡について

監査証跡の定義

監査証跡は、オプション指定の SAS ファイルです。このファイルを作成すると、SAS データファイルの修正情報をログに記録できます。オブザベーションが追加、削除、更新されるたびに、修正者、修正内容、修正日時に関する履歴情報が監査証跡ファイルに記録されます。

多くの企業や組織が、システムのセキュリティを確保するために監査証跡を必要としています。監査証跡は、データの履歴情報を保持します。この情報を使用することで、利用統計や利用パターンを解析できます。また、履歴情報を使用すると、データがデータファイルに入力された時点から削除された時点まで、個々のデータを追跡できます。

監査証跡は、追加に失敗したオブザベーションと、一貫性制約によって拒否されたオブザベーションを格納する場所でもあります。(一貫性制約の詳細については、“一貫性制約について” (619 ページ)を参照してください)。監査証跡を使用すると、追加に失敗または拒否されたオブザベーションを監査証跡ファイルから抽出し、失敗した理由を示す情報を利用してオブザベーションを修正し、マスタデータファイルに適用し直すような DATA ステッププログラムを記述できます。

監査証跡の説明

監査証跡は、デフォルト Base SAS Engine によって作成される SAS ファイルであり、データファイルと同一のライブラリ参照名とメンバ名を持ちますが、AUDIT タイプになります。監査証跡ファイルは、データファイル内にある変数をコピーして保持するのに加えて、次に示す 2 種類の監査変数を保持します。

- _AT*_変数(自動的に修正データを格納する変数)
- ユーザー変数(修正データの収集を定義できるオプション変数)

_AT*_変数についての説明を、次の表に示します。

表 26.2 _AT*_変数

_AT*_変数	説明
ATDATETIME	修正日時を格納します。

_AT*_変数	説明
ATUSERID	修正に関連付けられるユーザーのログオンユーザー ID を格納します。
ATOBSNO	修正の影響を受けるオブザベーション番号を格納します。ただし、REUSE=YES の場合は除きます(オブザベーション番号は常に 0 であるため)。
ATRETURNCODE	イベントのリターンコードを格納します。
ATMESSAGE	修正時に SAS ログに表示されたメッセージを格納します。
ATOPCODE	修正の内容を表す操作コードを格納します。

_ATOPCODE_の操作コードの値を次の表に示します。

表 26.3 _ATOPCODE_値

操作コード	修正内容
AL	監査再開
AS	監査中断
DA	データレコードイメージを追加
DD	データレコードイメージを削除
DR	レコードイメージの更新前処理
DW	レコードイメージの更新後処理
EA	オブザベーションの追加に失敗
ED	オブザベーションの削除に失敗
EU	オブザベーションの更新に失敗

対応する_ATOPCODE_値とともに監査証跡に格納されるエントリのタイプは、その監査証跡の開始時に LOG ステートメントで指定したオプションにより決定されます。監査証跡の開始時に LOG ステートメントを省略した場合、デフォルトの動作では、すべてのイメージがログに記録されます。

- A で始まる操作コードは、ADMIN_IMAGE オプションにより制御されます。
- 操作コード DR は、BEFORE_IMAGE オプションにより制御されます。
- D で始まる操作コードは、DATA_IMAGE オプションにより制御されます。
- E で始まる操作コードは、ERROR_IMAGE オプションにより制御されます。

ユーザー変数とは、データファイルに含められることなく、データファイルと関連付けられる変数のことです。すなわち、データ値は監査ファイルに格納されますが、ユーザー変数は他の変数と同様に、ユーザーによりデータファイル内で更新されます。ユーザ

一変数を定義することで、たとえば、ユーザーに個々の更新の理由を入力させるようにすることができます。

データファイルが持つことのできる監査証跡ファイルは 1 つだけであり、監査証跡ファイルは、監査するデータファイルと同一の SAS ライブラリに存在する必要があります。

ユーザー変数の定義と使用

ユーザー変数は、監査証跡の開始時に、USER_VAR ステートメントで定義します。たとえば、次のコードでは、監査証跡を開始した後、データファイル MyLib.Sales のユーザー変数 Reason_Code を定義しています。

```
proc datasets lib=mylib;
  audit sales;
     initiate;
     user_var reason_code $ 20;
run;
```

ユーザー変数を定義する場合は、変数が有効になるように値を格納する必要があります。プログラムを通じて、データ変数の場合と同じように、ユーザー変数にデータ値を入力できます。“データファイルの更新の例” (610 ページ)を参照してください。各オブザベーションが保存されるたびに、データ値が監査証跡ファイルに書き込まれます。ユーザー変数は、対話型のウィンドウ内では表示されず、更新もされません(SAS/FSP ソフトウェアの FSEDIT ウィンドウは除く)。監査変数を表示するには、TYPE=AUDIT データセットオプションを使用して監査ファイルを出力します。

ユーザー変数の名前の変更やその属性の修正を行うには、監査証跡ファイルではなく、データファイルを修正します。次の例では、PROC DATASETS を使用してユーザー変数の名前を変更しています。

```
proc datasets lib=mylib;
   modify sales;
     rename reason_code = Reason;
   run;
quit;
```

また、PROC DATASETS を使用して、データファイルにおける出力形式や入力形式などの属性を定義する必要があります。

共有環境での操作

監査証跡は、ローカル環境とリモート環境で同じように動作します。唯一の違いは、ユーザーやアプリケーションが SAS/CONNECT ソフトウェアおよび SAS/SHARE ソフトウェアを使用してネットワークに接続している場合には、オブザベーションが永久ストレージに書き出された時点で、監査証跡がイベントをログに記録することです。すなわち、データがリモート SAS セッションまたはサーバーに書き出された時点で、監査証跡はイベントをログに記録します。したがって、トランザクションがログに記録される時刻は、ユーザーの SAS セッションの時刻とは異なる場合があります。

パフォーマンスへの影響

データファイルに対する更新内容は監査証跡ファイルにも書き込まれるため、監査証跡の実行はシステムのパフォーマンスに悪影響を与える場合があります。定期的な大規模バッチ更新の場合、監査証跡ファイルへのイベントの記録を中断できます。ただし、監査証跡の実行を中断している間は、監査変数を利用できません。

他の操作による監査証跡の保持

監査証跡は、コピー、移動、並べ替え、置き換え、別の動作環境への転送などが行われるデータファイルに対しては推奨されません。これらの操作では、監査証跡を保持できないためです。同一ホスト上でコピー操作を実施する場合には、世代データセット機能を使用してデータファイルと監査証跡ファイルの名前を変更することにより、それらが失われないようにできます。ただし、監査処理も世代データセット機能も、置き換えを発生させたソースプログラムを保存することはできないため、ログへの記録は停止します。世代データセット機能に関する詳細については、“世代データセットについて” (613 ページ)を参照してください。

プログラミングの留意点

ユーザー変数を含む監査証跡ファイルに対応するデータファイルの変数リストは、データファイルの表示と更新とで異なります。ユーザー変数が選択されるのは更新のためであり、表示のためではありません。独自のフルスクリーンアプリケーションを開発する場合、この違いに留意する必要があります。

その他の留意点

削除操作では、ユーザー変数に入力されたデータ値は、監査証跡ファイルに格納されません。

監査証跡ファイルが破損した場合は、監査証跡を終了するまで、データファイルを処理できません。その後、新しい監査証跡を開始するか、監査証跡を使用しないでデータファイルを処理します。世代データセットの監査証跡を終了するには、AUDIT ステートメントの GENNUM=データセットオプションを使用します。この操作を行うと、世代データセットの監査証跡を開始できなくなります。

インデックス付きデータセットによる簡易追加機能を使用すると、いくつかのオブザベーションが監査証跡に重複して 2 回書き込まれる場合があります。その場合、1 回目は DA 操作コードで、2 回目は EA 操作コードで書き込まれます。EA 操作コードを含むオブザベーションは、インデックス制限によって拒否されたものであることを表します。詳細については、“Appending to an Indexed Data Set — Fast-Append Method” (*Base SAS Procedures Guide*)を参照してください。

監査証跡の初期化

監査証跡を初期化するには、DATASETS プロシジャで AUDIT ステートメントを使用します。構文については、“DATASETS” (*Base SAS Procedures Guide*)を参照してください。

監査証跡ファイルは、それが関連付けられているデータファイルに割り当てられている SAS パスワードを使用します。このため、データファイルには ALTER パスワードを設定することをお勧めします。ALTER パスワードは、SAS ファイルへの読み取りおよび編集に対するアクセスを制限します。ALTER パスワード以外を使用する場合やパスワードを使用しない場合、誤った更新または削除からファイルが保護されていない、という警告メッセージが表示されます。

監査証跡の制御

監査証跡を有効にした後、ログ記録の中断や再開、監査証跡の終了(削除)が行えます。監査証跡の制御に使用する構文については、DATASETS プロシジャの AUDIT

ステートメントの説明を参照してください。監査するデータファイルを置き換えると、監査証跡ファイルは削除されます。

監査証跡の読み取りとステータスの特定

監査証跡は読み取り専用です。データセットを読み取る SAS System の機能を使用すると、監査証跡ファイルを読み取ることができます。監査証跡ファイルを読み取るには、TYPE=データセットオプションを使用します。たとえば、監査証跡ファイルの内容を表示するには、次のステートメントを実行します。TYPE=オプションを丸かっこで囲む必要があることに注意してください。

```
proc contents data=mylib.sales (type=audit);
run;
```

上記の CONTENTS プロシジャの出力は次のとおりです。この出力には、対応するデータファイル内にあるすべての変数、_AT*_変数、ユーザー変数が含まれています。

アウトプット 26.1 *CONTENTS プロシジャによるデータファイル MyLib.Sales の出力*

The SAS System

The CONTENTS Procedure

Data Set Name	MYLIB.SALES.AUDIT	Observations	10
Member Type	AUDIT	Variables	10
Engine	V9	Indexes	0
Created	08/17/2012 10:31:41	Observation Length	263
Last Modified	08/17/2012 10:35:10	Deleted Observations	0
Protection		Compressed	NO
Data Set Type	AUDIT	Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	4096
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	14
Obs in First Data Page	10
Number of Data Set Repairs	0
Filename	C:\My Documents\sales.sas7baud
Release Created	9.0401B0
Host Created	W32_7PRO

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
5	_ATDATETIME_	Num	8	DATETIME19.
10	_ATMESSAGE_	Char	160	
6	_ATOBSNO_	Num	8	
9	_ATOPCODE_	Char	2	
7	_ATRETURNCODE_	Num	8	
8	_ATUSERID_	Char	32	
2	invoice	Num	8	
1	product	Char	9	
4	reason_code	Char	20	
3	renewal	Num	8	

REPORT プロシジャや TABULATE プロシジャなどを使用して、監査証跡に関するレポートを作成することもできます。

監査証跡とCEDA 処理

SAS データファイルで CEDA を使用した処理が必要となる場合、監査証跡はサポートされません。たとえば、Windows から UNIX のような別の動作環境に、初期化された監査証跡が付いた SAS データファイルを転送する場合、CEDA が対象ファイルを変換します。ただし、監査証跡は使用できません。CEDA 処理の詳細については、32 章, “クロス環境データアクセス(CEDA)を用いたデータ処理” (707 ページ)を参照してください。

MIGRATE プロシジャは、移行データセットに、削除オブザベーションをすべて保持します。そのため、MIGRATE プロシジャは監査証跡を保持し、移行します。詳細については、“MIGRATE” (*Base SAS Procedures Guide*)を参照してください。

一方、CPORT プロシジャや CIMPORT プロシジャなどの変換プロシジャはデータセットをクリーンアップして再構成します。たとえば、これらのプロシジャは削除されたオブザベーションを除去してディスク領域を回復します。再構成はこのように利点もありますが、監査証跡から変更を追跡しようとすると、データセットの履歴が正確でないことがあります。これらの変換プロシジャは削除されたオブザベーションを保持しないため、これらのプロシジャを使用して監査証跡をコピーすることはできません。詳細については、“CPORT” (*Base SAS Procedures Guide*) および “CIMPORT” (*Base SAS Procedures Guide*)を参照してください。

注意:
データファイルが監査証跡を含んでいる場合には、現在の動作環境のコマンドを使ってそのデータファイルのコピー、移動、削除を行わないでください。

監査証跡の使用例

監査証跡の初期化例

次の例では、データファイル MyLib.Sales の監査証跡の作成と初期化に使用されるデータとコードを紹介します。データファイル MyLib.Sales は、本セクションで前述した例で使用したものです。MyLib.Sales には、SAS プロダクトに関する架空の請求額と更新額が格納されています。監査証跡ファイルには、すべてのイベントが記録されるとともに、ユーザーの更新理由を入力するユーザー変数 Reason_Code が格納されます。

その次の例では、データファイルの更新が監査証跡に与える影響、および一貫性制約によって拒否されたオブザベーションを、監査変数を使用して読み取る方法を示します。

```
libname mylib 'C:\My Documents';
   /*-----------------------------------*/
   /* Create SALES data set.            */
   /*-----------------------------------*/

data mylib.sales;
  length product $9;
  input product invoice renewal;
datalines;
FSP         1270.00         570
SAS         1650.00         850
STAT        570.00          0
STAT        970.82          600
OR          239.36          0
SAS         7478.71         1100
```

```
SAS        800.00         800
;


   /*----------------------------------*/
   /* Create an audit trail with a     */
   /* user variable.                   */
   /*----------------------------------*/

proc datasets lib=mylib nolist;
  audit sales;
    initiate;
    user_var reason_code $ 20;
quit;
```

データファイルの更新の例

次の例では、データセット MyLib.Sales.Data にオブザベーションを挿入し、監査証跡ファイル MyLib.Sales.Audit の更新データを出力します。

```
/*----------------------------------*/
   /* Do an update.                    */
   /*----------------------------------*/
 proc sql;
   insert into mylib.sales
       set product = 'AUDIT',
           invoice = 2000,
           renewal = 970,
       reason_code = "Add new product";
quit;

   /*----------------------------------------*/
   /* Print the audit trail. */
   /*----------------------------------------*/
proc sql;
  select product,
         reason_code,
         _atopcode_,
         _atdatetime_
         from mylib.sales(type=audit);
quit;
```

アウトプット 26.2 *監査証跡ファイル MyLib.Sales.Audit 内の更新データ*

The SAS System

product	reason_code	_ATOPCODE_	_ATDATETIME_
AUDIT	Add new product	DA	10APR2014:14:35:23

監査証跡を使用して拒否されたオブザベーションをキャプチャする例

次の例では、データセット MyLib.Sales.Data に一貫性制約を追加し、一貫性制約によって拒否されたオブザベーションを監査証跡ファイル MyLib.Sales.Audit に記録しま

す。一貫性制約に関する詳細については、“一貫性制約について” (619 ページ)を参照してください。

```
   /*----------------------------------*/
   /* Create integrity constraints.    */
   /*----------------------------------*/
proc datasets lib=mylib;
   modify sales;
   ic create null_renewal = not null (invoice)
             message = "Invoice must have a value.";
   ic create invoice_amt = check (where=((invoice > 0) and
               (renewal <= invoice)))
             message = "Invoice and/or renewal are invalid.";
run;

   /*----------------------------------*/
   /* Do some updates.                 */
   /*----------------------------------*/
 proc sql; /* this update works */
    update mylib.sales
      set invoice = invoice * .9,
      reason_code = "10% price cut"
      where renewal > 800;

 proc sql;  /* this update fails */
    insert into mylib.sales
       set product = 'AUDIT',
           renewal = 970,
       reason_code = "Add new product";

 proc sql;  /* this update works */
    insert into mylib.sales
       set product = 'AUDIT',
           invoice = 10000,
           renewal = 970,
       reason_code = "Add new product";

proc sql;  /* this update fails */
    insert into mylib.sales
       set product = 'AUDIT',
           invoice = 100,
           renewal = 970,
       reason_code = "Add new product";
 quit;

   /*----------------------------------------*/
   /* Print the audit trail. */
   /*----------------------------------------*/
proc print data=mylib.sales(type=audit);
  format _atuserid_ $6.;
  var product reason_code _atopcode_ _atdatetime_;
title  'Contents of the Audit Trail';
run;

   /*----------------------------------------*/
```

```
   /* Print the rejected records.            */
   /*---------------------------------------*/
proc print data=mylib.sales(type=audit);
  where _atopcode_ eq "EA";
  format _atmessage_ $250.;
  var product invoice renewal _atmessage_ ;
title  'Rejected Records';
run;
```

次の出力は、MyLib.Sales.Data が何度か更新された後の MyLib.Sales.Audit の内容を示しています。一貫性制約がファイルに追加され、更新が行われました。

アウトプット 26.3 *更新後の監査証跡ファイル MyLib.Sales.Audit の内容(一貫性制約あり)*

Contents of the Audit Trail

Obs	product	reason_code	_ATOPCODE_	_ATDATETIME_
1	AUDIT	Add new product	DA	10APR2014:14:35:23
2	SAS		DR	10APR2014:14:37:13
3	SAS	10% price cut	DW	10APR2014:14:37:13
4	SAS		DR	10APR2014:14:37:13
5	SAS	10% price cut	DW	10APR2014:14:37:13
6	AUDIT		DR	10APR2014:14:37:13
7	AUDIT	10% price cut	DW	10APR2014:14:37:13
8	AUDIT	Add new product	EA	10APR2014:14:37:13
9	AUDIT	Add new product	DA	10APR2014:14:37:13
10	AUDIT	Add new product	EA	10APR2014:14:37:13

この出力は、拒否されたオブザベーションに関する、監査証跡にある情報を示しています。

アウトプット 26.4 *監査証跡上の拒否されたレコード*

Rejected Records

Obs	product	invoice	renewal	_ATMESSAGE_
1	AUDIT	.	970	ERROR: Invoice must have a value. Add/Update failed for data set MYLIB.SALES because data value(s) do not comply with integrity constraint null_renewal.
2	AUDIT	100	970	ERROR: Invoice and/or renewal are invalid. Add/Update failed for data set MYLIB.SALES because data value(s) do not comply with integrity constraint invoice_amt.

世代データセットについて

世代データセットの定義

世代データセットとは、世代グループの一部として格納される SAS データセットのアーカイブ化されたバージョンのことです。世代データセットは、ファイルが置き換えられるたびに作成されます。1 つの世代グループ内の各世代データセットは同じルートメンバ名を持ちますが、バージョン番号はそれぞれ異なっています。最も新しいバージョンの世代データセットのことを、ベースバージョンと呼びます。

世代を管理できるのは SAS データファイルに関してのみです。SAS ビューに関しては世代を管理できません。

注: 世代データセットは、データセットの履歴バージョンを提供します。これは、データセットの個々のオブザベーションの更新の履歴ではありません。オブザベーションの追加、削除、更新が行われるたびにログを記録する方法については、“監査証跡について” (603 ページ)を参照してください。

注意:

世代データセットの管理には、オペレーティングシステムツールを使用しないでください。これが、世代グループファイルへのアクセスを制限することになる場合があります。

世代データセット関連の用語

世代データセットに関連する用語を次に示します。

ベースバージョン
: 最後に作成された最も新しいバージョンのデータセットです。ベースバージョン名には、世代番号を示す 4 文字の接尾辞が付いていません。

世代グループ
: 元のデータセットに対する置き換えの履歴を表すデータセットのグループです。世代グループは、ベースバージョンと一連の履歴バージョンで構成されます。

世代番号
: 世代グループで、履歴バージョンの特定の 1 つの世代(バージョン)を示す数で、ルートメンバ名の後ろに付き、1 ずつ増加します。たとえば、Air#272 という名前のデータセットの世代番号は 272 です。

GENMAX=
: データセットの世代を要求し、任意のデータセットで保持するバージョン(ベースバージョンと履歴バージョンを含む)の最大数を指定する出力データセットオプションです。デフォルト値は GENMAX=0 であり、これは世代データセット機能が無効であることを意味します。

GENNUM=
: 世代グループに含まれている特定のバージョンを参照する入力データセットオプションです。正の数は、世代番号に基づく履歴バージョンへの絶対参照となります。負の数は、履歴バージョンへの相対参照となります。たとえば、GENNUM=-1 と指定すると最も若いバージョンを参照します。

履歴バージョン
: 以前のベースバージョンの履歴です。履歴バージョンの名前には、#003 など、世代番号を表す 4 文字の接尾辞が付いています。

最も古いバージョン
世代グループのうちで、最も古いバージョンのデータセットです。

ロールオーバー
バージョン番号が 999 から 000 に移行するプロセスのことです。世代番号が 999 に達すると、次の値は 000 になります。

最も若いバージョン
世代番号がベースバージョンに最も近いバージョンです。

世代データセットの呼び出し

世代データセットを呼び出す場合や、保持できるバージョンの最大数を指定する場合には、データセットの作成または置換時に GENMAX=出力データセットオプションを指定します。たとえば、次の DATA ステップでは、新しいデータセットを作成するとともに、履歴を 4 世代まで保持するように指示しています。4 世代とは、1 つのベースバージョンと 3 つの履歴バージョンです。

```
data (genmax=4);
   x=1;
   output;
run;
```

GENMAX=データセットオプションを有効にすると、データセットメンバ名の最大長は (32 文字ではなく)28 文字までに制限されます。これは、末尾の 4 文字がバージョン番号用に予約されるためです。GENMAX=データセットオプションを無効にすると、データセットメンバ名の最大長は 32 文字になります。詳細については、*SAS データセットオプション: リファレンス*にある GENMAX=データセットオプションの説明を参照してください。

世代グループのメンテナンス方法について

世代管理が有効なデータセットが初めて置き換えられると、SAS System は置き換えられたデータセットを保持したまま、そのメンバ名に 4 文字のバージョン番号を追加します。このバージョン番号は、#と 3 桁の番号から構成されます。A という名前のデータセットがある場合、置き換えられたデータセットは A#001 になります。次にデータセットが置き換えられると、置き換えられたデータセットは A#002 になります。つまり、A#002 がベースバージョンに時間的に最も近い世代になります。3 回の置き換えが実行された後は、次のような結果になります。

A
ベース(現在の)バージョン

A#003
最も新しい(最も若い)履歴バージョン

A#002
2 番目に新しい履歴バージョン

A#001
最も古い履歴バージョン

GENMAX=4 と指定した場合、4 回目の置き換えが実行されると、最も古いバージョンである A#001 が削除されます。置き換えが実行されると、常に履歴が 4 世代保持されます。たとえば、10 回の置き換え後には次のような結果になります。

A
ベース(現在の)バージョン

A#010
最も新しい(最も若い)履歴バージョン

A#009
2 番目に新しい履歴バージョン

A#008
最も古い履歴バージョン

SAS System が追加できるバージョン番号の上限は、#999 です。つまり、999 回の置き換え実行後には、最も若いバージョンは#999 になります。1,000 回の置き換え後には、最も若いバージョン番号が#000 にロールオーバーされます。1,001 回の置き換え後には、最も若いバージョン番号は#001 になります。たとえば、データセット A を使用するときに GENNUM=4 と指定すると、結果は次のようになります。

999 回の置き換え
- A (現在)
- A#999 (最も新しい)
- A#998 (2 番目に新しい)
- A#997 (最も古い)

1,000 回の置き換え
- A (現在)
- A#000 (最も新しい)
- A#999 (2 番目に新しい)
- A#998 (最も古い)

1,001 回の置き換え
- A (現在)
- A#001 (最も新しい)
- A#000 (2 番目に新しい)
- A#999 (最も古い)

世代グループの命名規則を次の表に示します。

表 26.4 *世代グループデータセットの命名規則*

時間	SAS コード	データセット名	GENNUM=絶対参照	GENNUM=相対参照	説明
1	data air (genmax=3);	Air	1	0	1 回目で Air データセットが作成され、3 世代の世代管理が要求されます。
2	data air;	Air Air#001	2 1	0 -1	Air が置き換えられます。1 回目の Air の名前は Air#001 に変更されます。
3	data air;	Air Air#002 Air#001	3 2 1	0 -1 -2	Air が置き換えられます。2 回目の Air の名前は Air#002 に変更されます。

時間	SAS コード	データセット名	GENNUM=絶対参照	GENNUM=相対参照	説明
4	data air;	Air Air#003 Air#002	4 3 2	0 -1 -2	Air が置き換えられます。3 回目の Air の名前は Air#003 に変更されます。最も古い 1 回目の Air#001 は削除されます。
5	data air (genmax=2);	Air Air#004	5 4	0 -1	Air は置き替えられ、世代の数が 2 に変更されます。4 回目の Air の名前は Air#004 に変更されます。バージョンが最も古い方から 2 つ削除されます。

世代グループの特定バージョンの処理

世代グループが存在している場合、デフォルトではベースバージョンが処理されます。たとえば、次の PRINT プロシジャを実行するとベースバージョンが出力されます。

```
proc print data=a;
run;
```

世代グループの特定のバージョンを要求するには、GENNUM=入力データセットオプションを使用します。次の 2 つの方法があります。

- 0 以外の正の整数を指定すると、特定の履歴バージョン番号を絶対参照します。たとえば、次のステートメントを実行すると履歴バージョン#003 が出力されます。

  ```
  proc print data=a(gennum=3);
  run;
  ```

  注: 1,000 回の置き換え後に履歴バージョン#000 を取得したい場合は、GENNUM=1000 を指定します。

- 負の整数を指定すると、ベースバージョンを起点とした、最も若いバージョンから最も古いバージョンへの相対参照になります。たとえば、GENNUM=-1 と指定すると最も若いバージョンを参照します。次のステートメントは、ベースバージョンから 3 バージョン前のデータセットを出力します。

  ```
  proc print data=a(gennum=-3);
  run;
  ```

表 26.5 *特定の世代データセットの要求*

SAS ステートメント	結果
`proc print data=air (gennum=0);` `proc print data=air;`	Air データセットの現在の(ベース)バージョンを出力します。
`proc print data=air (gennum=-2);`	現在の世代から 2 世代前のバージョンを出力します。
`proc print data=air (gennum=3);`	ファイル Air#003 を出力します。
`proc print data=air (gennum=1000);`	1,000 回の置き換えが実行された後、Air#999 の次に作成されたファイル Air#000 を出力します。

世代グループの管理

紹介

DATASETS プロシジャは、世代グループを管理するための各種ステートメントを提供します。DATASETS プロシジャでは、GENNUM=オプションは次のような追加機能を持つことに注意してください。

- DATASETS プロシジャの DELETE ステートメントでは、GENNUM=オプションは追加の値として ALL、HIST、REVERT をサポートします。
- CHANGE ステートメントでは、GENNUM=オプションは追加の値として ALL をサポートします。
- CHANGE ステートメントでは、GENNUM=0 を指定することは、ベースバージョンではなく、すべてのバージョンを意味します。

注意:

世代データセットの管理には、オペレーティングシステムツールを使用しないでください。これが、世代グループファイルへのアクセスを制限することになる場合があります。そのかわりに、DATASETS または COPY プロシジャなどの SAS ツールを使用します。

データセット情報の表示

DATASETS プロシジャのさまざまなステートメントを使用すると、特定の履歴バージョンを処理できます。たとえば、DATASETS プロシジャの CONTENTS ステートメントを使用すると、履歴コピーのデータセットバージョン番号を表示できます。

```
proc datasets library=myfiles;
   contents data=test (gennum=2);
run;
```

世代グループのコピー

DATASETS プロシジャの COPY ステートメントを使用するか、または COPY プロシジャを使用すると、世代グループをコピーできます。ただし、個別のバージョンはコピーできません。

たとえば、次の DATASETS プロシジャでは、COPY ステートメントを使用して、データセット MyGen1 の世代グループをライブラリ MyLib1 からライブラリ MyLib2 にコピーしています。

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

proc datasets;
  copy in=mylib1 out=mylib2;
  select mygen1;
run;
```

世代グループの追加

GENNUM=データセットオプションを使用すると、特定の履歴バージョンを追加できます。たとえば、次の例では、DATASETS プロシジャの APPEND ステートメントを使用して、データセット B の履歴バージョンをデータセット A に追加しています。デフォルトでは、SAS System は BASE=オプションに指定されたデータセットをベースバージョンとして使用することに注意してください。

```
proc datasets;
```

```
   append base=a data=b(gennum=2);
run;
```

バージョン数の変更

データセットの属性を変更する場合、既存の世代グループに含まれるバージョンの数を増減できます。

たとえば、次の DATASETS プロシジャの MODIFY ステートメントは、データセット MyLib.Air の世代数を 4 に変更します。

```
libname mylib 'SAS-library';

proc datasets library=mylib;
   modify air(genmax=4);
run;
```

注意:

バージョン数を減らすと、SAS System は、世代の古いものから順番に履歴バージョンを削除して、新しい最大数を超えないようにします。 たとえば、次の MODIFY ステートメントでは、MyLib.Air のバージョン数を 4 から 0 に減らしています。この結果、SAS System は 3 つの履歴バージョンを自動的に削除します。

```
proc datasets library=mylib;
   modify air (genmax=0);
run;
```

世代グループのバージョンの削除

データセットを削除する場合、特定のバージョンを削除することも、世代グループ全体を削除することもできます。次の表は、削除操作の種類と、世代グループのバージョンを削除したときの影響を示しています。

次の例では、ベースバージョンの Air と 2 つの履歴バージョン Air#001 と Air#002 が存在している状態に、それぞれのコマンドを発行すると想定します。

表 26.6 世代データセットの削除

DATASETS プロシジャ内の SAS ステートメント	結果
`delete air;` `delete air(gennum=0);`	ベースバージョンが削除され、履歴バージョンがシフトアップされます。Air#002 が Air という名前に変更され、新しいベースバージョンになります。
`delete air(gennum=2);`	履歴バージョン Air#002 が削除されます。
`delete air(gennum=-2);`	2 番目に若い履歴バージョン(Air#001)が削除されます。
`delete air(gennum=all);`	世代グループに含まれるデータセットが、ベースバージョンを含め、すべて削除されます。
`delete air(gennum=hist);`	世代グループに含まれるデータセットが、ベースバージョンを除いてすべて削除されます。

注: 絶対参照と相対参照の両者はともに特定のバージョンを参照します。相対参照は削除済みのバージョンをスキップしません。このため、1 つまたは複数の削除済みバージョンを含んでいる世代グループを処理する場合、参照先のバージョンが削

除されているならば、相対参照を使用するとエラーが発生します。たとえば、ベースバージョン Air と 3 つの履歴バージョン(Air#001、Air#002、Air#003)が存在する場合に、Air#002 を削除したとします。この場合、次のステートメントを実行すると、Air#002 が存在しないため、エラーが返されます。SAS System は、GENNUM=-2 というオプションでユーザーが Air#003 を意図しているとは推測しません。

```
proc print data=air (gennum= -2);
run;
```

世代グループのバージョン名の変更

データセット名を変更する場合、次のように指定すると世代グループ全体の名前を変更できます。

```
proc datasets;
   change a=newa;
run;
```

単一の履歴バージョンの名前を変更するには、次に示すように GENNUM=オプションを含めます。

```
proc datasets;
   change a(gennum=2)=newa;
```

注: DATASETS プロシジャの CHANGE ステートメントで GENNUM=0 を指定すると、世代グループ全体を変更できます。

世代グループでのパスワードの使用

世代グループ内のバージョンに割り当てたパスワードは、次のように管理されます。

- ベースバージョンにパスワードを割り当てた場合、そのパスワードはそれ以降に作成された履歴バージョンで保持されます。そのパスワードは、既存の履歴バージョンには適用されません。
- ある履歴バージョンにパスワードを割り当てた場合、そのパスワードはその個別データセットのみに適用されます。

一貫性制約について

一貫性制約の定義

一貫性制約とはデータ検証規則のセットです。これを指定することで、SAS データファイルの変数に格納できる有効なデータ値をユーザーが制限できます。一貫性制約を指定すると、格納データの有効性と整合性を管理できます。SAS System は、一貫性制約が割り当てられた変数の値が追加、更新、または削除されるたびに、一貫性制約を適用します。

一貫性制約には、汎用制約と参照制約の 2 種類があります。

注意:

一貫性制約の管理にオペレーティングシステムツールは使用しないでください。これによって、データセットが破損する場合があります。そのかわりに、DATASETS プロシジャや SQL プロシジャなどの SAS ツールを使用します。

汎用一貫性制約と参照一貫性制約

汎用一貫性制約

汎用一貫性制約を使用すると、1 つのファイル内の変数値を制限できます。汎用制約には次の 4 種類があります。

CHECK
変数のデータ値を、指定された集合、範囲、リストに制限します。また、CHECK 制約を使用すると、1 つのオブザベーション内の 1 つの変数のデータ値が、同じオブザベーション内の別のデータ値に依存することを確認できます。

NOT NULL
変数の値に欠損値を含めないことを指定します。Null 値(欠損値)は許可されません。

UNIQUE
変数の値が一意であることを指定します。ヌルデータ値は許可されますが、単一のインスタンスに制限されます。

主キー
変数の値は一意であり、ヌルデータ値を含めないことを指定します。1 つのデータファイルには、主キーが 1 つだけ存在できます。

注: データファイルがそれ自身を参照する外部キー制約を持たない場合、汎用一貫性制約が主キーとなります。

参照一貫性制約

参照一貫性制約は、あるデータファイル内に存在する主キーの一貫性制約が、別のデータファイル内に存在する外部キーの一貫性制約によって参照される場合に作成されます。

外部キー制約は、外部キーデータファイル内にある 1 つ以上の変数のデータ値を、主キーデータファイル内にある対応する変数や値へリンクします。外部キーデータファイル内のデータ値は、主キーデータファイル内の値と一致するか、または Null でなければなりません。主キーデータファイル内でデータが更新または削除された場合、そのような変更は、外部キー制約の一部として定義されている参照アクションにより制御されます。

独立した参照アクションを、更新および削除の各操作に関して定義できます。参照アクションには次の 3 つの種類があります。

RESTRICT
主キー変数のデータ値が、外部キーデータファイルの対応する外部キー変数のいずれかの値と一致する場合に、更新または削除されないようにします。これは、参照アクションが指定されていない場合のデフォルトです。

SET NULL
主キー変数のデータ値の更新または削除を許可し、外部キーデータファイル内の一致するデータ値を Null 値(欠損値)へと変更します。

CASCADE
主キー変数のデータ値の更新を可能にし、外部キーデータファイル内の一致するデータ値を同じ値へと更新します。このアクションは、更新操作でのみサポートされます。

参照制約を確立するには、次の条件を満たす必要があります。

- 主キーと外部キーが同じ数の変数を参照していること、およびそれらの変数が同じ順序を持つこと。

- 変数の種類(文字または数値)および長さが同じであること。
- すでにデータを含んでいるデータファイルに対して外部キーを追加する場合、外部キーデータファイル内のデータ値は、主キーの既存の値と一致するか、または Null であること。

外部キーデータファイルは、参照先の主キーデータファイルと同じ SAS ライブラリ内か、または別の SAS ライブラリ内に存在できます(前者をライブラリ参照名内の参照制約、後者をライブラリ参照名間の参照制約と呼びます)。ただし、外部キーデータファイルを含んでいるライブラリが一時ライブラリである場合、主キーデータファイルを含んでいるライブラリも同様に一時ライブラリでなければなりません。また、参照一貫性制約を、連結ライブラリ内のデータファイルに割り当てることはできません。

主キーを参照できる外部キーの数には制限がありません。ただし、あまりに多くの外部キーを追加すると、更新および削除操作に対して悪影響を与える場合があります。

参照制約が存在する場合、主キーの一貫性制約は、それを参照しているすべての外部キーが削除されるまで削除されません。外部キーの削除に対する制限はありません。

主キー制約と外部キー制約の重複

SAS データファイル内の変数は、主キー(汎用一貫性制約)および外部キー(参照一貫性制約)の両方に含めることができます。ただし、同じ変数を使用する主キーと外部キーを定義する場合、次の制限が適用されます。

- 外部キーの更新および削除に対して定義される参照アクションは、どちらも RESTRICT であることが必要です。
- 同じ変数を主キーと外部キーの定義で使用する場合、それらの変数を異なる順序で定義する必要があります。

例については、“主キー制約と外部キー制約の重複の定義” (631 ページ)を参照してください。

一貫性制約の保持

次のプロシジャを使用して元のデータファイルをコピーすると、一貫性制約が保持されます。

- Base SAS ソフトウェアの APPEND、COPY、CPORT、CIMPORT、MIGRATE、SORT プロシジャ
- SAS/CONNECT ソフトウェアの UPLOAD および DOWNLOAD プロシジャ
- APPEND プロシジャ
  - 既存の BASE=データファイルでは、BASE=ファイル内の一貫性制約は保持されますが、その BASE=ファイルに追加される DATA=ファイルの一貫性制約は保持されません。
  - 存在しない BASE=データファイルの場合、新しい BASE=ファイルに追加される DATA=ファイル内の汎用一貫性制約が保持されます。DATA=ファイル内の参照一貫性制約は保持されません。
- SORT、UPLOAD、DOWNLOAD プロシジャ(OUT=データファイルが指定されていない場合)
- SAS **エクスプローラ**ウィンドウ

また、CONSTRAINT=オプションを使用しても、COPY、CPORT、CIMPORT、UPLOAD、DOWNLOAD プロシジャに対して、一貫性制約を保持するかどうかを制御できます。

汎用一貫性制約はアクティブ状態で保持されます。参照制約の状態が保持されるかどうかは、プロシジャが主キーと外部キーの各データファイルを同じ SAS ライブラリに書き出すか、それとも別の SAS ライブラリに書き出すかによって(すなわち、ライブラリ参照名内の一貫性制約であるか、それともライブラリ参照名間の一貫性制約であるかによって)決定されます。ライブラリ参照名内の参照制約は、アクティブ状態で保持されます。ライブラリ参照名間の参照制約は、非アクティブ状態で保持されます。つまり、一貫性制約の主キー部分は汎用制約として適用されますが、外部キー部分はアクティブではありません。アクティブでない外部キーを再びアクティブにするには、DATASETS プロシジャの IC REACTIVATE ステートメントを使用する必要があります。

次の表に、一貫性制約が保持される状況を示します。

表 26.7 *一貫性制約が保持される状況*

プロシジャ	条件	保持される制約
APPEND	DATA=オプションが指定されていない	汎用制約 参照制約は影響を受けません
COPY	CONSTRAINT=yes	汎用制約 アクティブ状態でのライブラリ参照名内の参照制約 非アクティブ状態でのライブラリ参照名間の参照制約
CPORT/CIMPORT	CONSTRAINT=yes	汎用制約 アクティブ状態でのライブラリ参照名内の参照制約 非アクティブ状態でのライブラリ参照名間の参照制約
SORT	OUT=オプションが指定されていない	汎用制約 参照制約は影響を受けません
UPLOAD/DOWNLOAD	CONSTRAINT=yes および OUT=オプションが指定されていない	汎用制約 アクティブ状態でのライブラリ参照名内の参照制約 非アクティブ状態でのライブラリ参照名間の参照制約
SAS エクスプローラウィンドウ		汎用制約

注意:

一貫性制約の管理にオペレーティングシステムツールは使用しないでください。これによって、データセットが破損する場合があります。そのかわりに、DATASETS プロシジャや SQL プロシジャなどの SAS ツールを使用します。

インデックスと一貫性制約

UNIQUE 制約(指定した変数で値が重複することを許可しない制約)、主キー(PRIMARY KEY)制約、外部キー一貫性制約は、データ値をインデックスファイルに格納します。インデックスファイルが存在する場合は、そのインデックスが使用されます。インデックスファイルが存在しない場合には、作成されます。一貫性制約を作成または削除するときは、次の点を考慮します。

- ユーザー定義のインデックスが存在する場合に、一貫性制約が作成されるためには、インデックスの属性に一貫性制約との互換性を持たせる必要があります。たとえば、主キー一貫性制約を追加する場合、既存のインデックスが UNIQUE オプションを指定して作成されたインデックス(重複した値を持たない)である必要があります。外部キー一貫性制約を追加する場合、インデックスが UNIQUE オプションを指定して作成したインデックスであってはいけません。
- UNIQUE 一貫性制約には UNIQUE インデックス属性と同じ効果があります。このため、両方を使用する必要はなく、どちらか一方だけを使用します。
- NOMISS インデックス属性と NOT NULL 一貫性制約の機能は類似していますが、効果は異なります。NOT NULL 一貫性制約は、SAS データファイルに欠損値が含まれることを防ぎます。欠損値を含む既存のデータファイルに、NOT NULL 制約を追加することはできません。NOMISS オプションを指定したインデックスは、データファイルに欠損値が含まれることを許可しますが、インデックスからは欠損値を削除します。
- インデックスを作成すると、インデックスは、ユーザー、一貫性制約、またはその両方によって所有されている状態となります。一貫性制約によって所有されているインデックスをユーザーが削除すること、およびユーザーによって所有されているインデックスを一貫性制約で削除することはできません。一貫性制約とユーザーの両方によって所有されているインデックスを削除するには、両方がインデックスを削除する必要があります。インデックスを削除できなかった場合は、SAS ログに情報が表示されます。

一貫性制約のロック

一貫性制約は、メンバレベルのロックおよびレコードレベルのロックの両方をサポートします。デフォルトのロックのレベルを上書きするには、CNTLLEV=データセットオプションを使用します。詳細については、“CNTLLEV= Data Set Option” (*SAS Data Set Options: Reference*)を参照してください。

暗号化と一貫性制約

SAS では、暗号化には、2 種類のアルゴリズムが使用されます。

- SAS 独自の暗号化は、ENCRYPT=YES データセットオプションで実装されます。
- AES (高度暗号化標準)暗号化は、ENCRYPT=AES データセットオプションで実装されます。

一貫性制約を使用する場合に、SAS 独自の暗号化には制限はありません。

AES 暗号化では、すべての主キーデータファイルと外部キーデータファイルで同一の暗号化キーを使用することが必要で、これにより、参照している外部キーデータファイルと主キーデータファイルのすべてがオープンされます。ENCRYPT=AES を使用する場合は、ENCRYPTKEY=データセットオプションを指定する必要があります。詳細については、“ENCRYPT= Data Set Option” (*SAS Data Set Options: Reference*) および

"ENCRYPTKEY= Data Set Option" (*SAS Data Set Options: Reference*)を参照してください。

暗号化キーがメタデータ連結ライブラリに記録されていない場合、暗号化キーは、主キーデータファイルと参照している暗号化外部キーデータファイルとで同じでなければなりません。メタデータ連結ライブラリの詳細については、"Metadata-Bound Library" (*Base SAS Procedures Guide*)を参照してください。

一貫性制約の指定

一貫性制約は、SQL プロシジャ、DATASETS プロシジャ、SCL (SAS コンポーネント言語)で作成します。データファイルを作成するときに一貫性制約を指定できます。また、既存の SAS データファイルに追加することもできます。一貫性制約を既存のデータファイルに追加する場合、SAS System は、一貫性制約を追加する前に、一貫性制約を割り当てる既存のデータ値が制約に従っていることを確認します。

一貫性制約を指定する場合は、各制約につき別々のステートメントを指定する必要があります。また、NOT NULL 一貫性制約を割り当てる変数ごとに、別々のステートメントを指定する必要があります。主キー制約、外部キー制約、UNIQUE 制約に複数の変数を指定すると、複合インデックスが作成され、一貫性制約が変数の値の組み合わせに適用されます。SAS インデックスと一貫性制約との関係については、"インデックスと一貫性制約" (654 ページ)を参照してください。詳細については、"SAS インデックスについて" (632 ページ)を参照してください。

SCL で一貫性制約を追加する場合は、データセットをユーティリティモードで開きます。例については、"SCL を使用した一貫性制約の作成" (627 ページ) を参照してください。一貫性制約の削除は、ユーティリティモードで行う必要があります。構文の詳細については、*SAS Component Language:Reference* を参照してください。

世代データセットを使用する場合は、保護された変数を含む世代データセットごとに一貫性制約を作成する必要があります。

注意:

SAS 9.2 の CHECK 制約は、それ以前のバージョンの SAS System とは互換性がありません。 CHECK 制約を既存の SAS データセットに割り当てた場合、または CHECK 制約を含む SAS データセットを作成した場合、そのデータセットは、SAS 9.2 より前のバージョンの SAS System からはアクセスできません。

ディスク領域の共有時にライブラリ参照名間の参照一貫性制約に物理場所を指定する

ネットワーク経由でディスク領域を共有しており、外部キーデータファイルと主キーデータファイルが異なる SAS ライブラリに存在するような参照一貫性制約にアクセスする場合、共有ファイルの物理的な保存場所に関する標準を確立する必要があります。共有ファイルを作成する場合、複数のネットワークマシンがその共有ファイルに同じ物理名を使用してアクセスできるようにするには、標準が必要となります。物理名が一致しない場合、SAS System は、参照されている外部キーデータファイルや主キーデータファイルをオープンできません。

たとえば、すべての共有ファイルをディスク T:上に配置するような標準を確立した場合、複数のネットワークマシンがその共有ファイルに同じパス名を使用してアクセスできるようになります。

標準なしにファイルを作成した場合に問題が発生する例を次に示します。たとえば、主キーデータファイルと外部キーデータファイルをマシン D4064 上の異なるディレクトリ `C:\Public\pkey_directory` および `C:\Public\fkey_directory` にそれぞれ作成したとします。これらのパス名は、SAS データファイルのディスクリプタ部に格納されます。

別のマシン(例: F2760)から主キーデータファイルにアクセスするには、次の LIBNAME ステートメントを実行します。

```
libname pkds '\\D4064\Public\pkey_directory';
```

主キーデータファイルを更新処理用にオープンする場合、SAS System は自動的に、主キーデータファイル内に格納されている外部キーデータファイルの物理名である **c:\Public\fkey_directory** を使用して、その外部キーデータファイルをオープンしようとします。ところが、そのようなディレクトリはマシン F2760 上には存在しません。そのため、外部キーデータファイルのオープンは失敗します。

一貫性制約のリスト表示

PROC CONTENTS および PROC DATASETS を使うと、特別なオプションを使用せずに、一貫性制約に関する情報を表示できます。また、OUT2=オプションを使用すると、一貫性制約とインデックスに関する情報を表示できます。SQL プロシジャの場合は、DESCRIBE TABLE ステートメントが一貫性制約の指定をデータファイル定義の一部として表示し、DESCRIBE TABLE CONSTRAINTS ステートメントが一貫性制約の指定を単独で表示します。SCL の場合は、ICTYPE 関数、ICVALUE 関数、ICDESCRIBE 関数が一貫性制約の情報を取得します。詳細については、*Base SAS Procedures Guide* と *SAS Component Language:Reference* を参照してください。

拒否されたオブザベーション

一貫性制約の作成時に MESSAGE=および MSGTYPE=オプションを使用すると、一貫性制約に関するエラーメッセージをカスタマイズできます。MESSAGE=オプションを使うと、ユーザー定義メッセージを、一貫性制約に関するエラーメッセージの先頭に付加することができます。MSGTYPE=オプションを使用すると、メッセージの SAS 部分を抑制できます。詳細については、DATASETS プロシジャ、SQL プロシジャ、および SCL の説明を参照してください。

監査証跡機能を使用すると、拒否されたオブザベーション情報を監査証跡ファイルに保存できます。

一貫性制約と CEDA 処理

SAS データファイルで CEDA を使用した処理が必要となる場合、一貫性制約はサポートされません。たとえば、初期化した一貫性制約を持つ SAS データファイルを Windows 環境から UNIX 環境へと転送する場合、CEDA はユーザーのためにファイルを自動的に変換しますが、一貫性制約は使用できません。CEDA 処理の詳細については、32 章, “クロス環境データアクセス(CEDA)を用いたデータ処理” (707 ページ)を参照してください。

MIGRATE プロシジャは、データファイルの移行時に一貫性制約を保持します。詳細については、“MIGRATE” (*Base SAS Procedures Guide*)を参照してください。CPORT プロシジャおよび CIMPORT プロシジャは、SAS データファイルをある動作環境から別の動作環境へと移送する場合に、一貫性制約を保持します。CPORT プロシジャは、移送可能な形式でデータファイルのコピーを作成します。CIMPORT プロシジャは、移送ファイルを読み込み、そのデータファイルのコピーをホスト固有の形式で新しく作成します。詳細については、“CPORT” (*Base SAS Procedures Guide*) と “CIMPORT” (*Base SAS Procedures Guide*)を参照してください。

例

DATASETS プロシジャを使用した一貫性制約の作成

次の例では、DATASETS プロシジャを使用して一貫性制約を作成します。データファイル TV_Survey では、次の一貫性制約を使用して、ネットワーク、PBS、その他のチャネルの視聴率を確認します。

- 視聴率は 100%を超えることはできない
- 調査対象は大人のみ
- 性別が男または女

```
data tv_survey(label='Validity checking');
   length idnum age 4 gender $1;
   input idnum gender age network pbs other;
datalines;
1 M 55 80 . 20
2 F 36 50 40 10
3 M 42 20 5 75
4 F 18 30 0 70
5 F 84 0 100 0
;

proc datasets nolist;
  modify tv_survey;
    ic create val_gender = check(where=(gender in ('M','F')))
       message = "Valid values for variable GENDER are
       either 'M' or 'F'.";
    ic create val_age = check(where=(age >= 18 and age = 120))
       message = "An invalid AGE has been provided.";
    ic create val_new = check(where=(network = 100));
    ic create val_pbs = check(where=(pbs = 100));
    ic create val_ot = check(where=(other = 100));
    ic create val_max = check(where=((network+pbs+other)= 100));
quit;
```

SQL プロシジャを使用した一貫性制約の作成

次の例では、SQL プロシジャを使用して一貫性制約を作成します。データファイル People は、従業員の一覧と従業員情報を含みます。データファイル Salary は、給与とボーナスの情報を含みます。一貫性制約は次のとおりです。

- ボーナスを受け取る従業員の名前がデータファイル People に存在すること
- 主キー変数で識別される名前(name)は一意(UNIQUE)であること
- 性別が男または女
- 仕事のステータス(status)は、終身(permanent)、臨時(temporary)、解雇(terminated)

```
proc sql;
   create table people
    (
     name      char(14),
     gender    char(6),
     hired     num,
     jobtype   char(1) not null,
```

```
    status    char(10),

   constraint prim_key primary key(name),
   constraint gender check(gender in ('male' 'female')),
   constraint status check(status in ('permanent'
                          'temporary' 'terminated'))
    );

    create table salary
    (
     name     char(14),
     salary   num not null,
     bonus    num,

     constraint for_key foreign key(name) references people
        on delete restrict on update set null
    );
  quit;
```

SCL を使用した一貫性制約の作成

SCL を使用してデータファイルに一貫性制約を追加するには、SCL カタログエントリを作成して構築する必要があります。次の例では、カタログエントリ Example.Ic_Cat_Allics.SCL を作成してコンパイルします。

```
INIT:
  put "Test SCL integrity constraint functions start.";
return;

MAIN:
   put "Opening WORK.ONE in utility mode.";
   dsid = open('work.one', 'V');/* Utility mode.*/
   if (dsid = 0) then
     do;
      _msg_=sysmsg();
      put _msg_=;
     end;
     else do;
      if (dsid > 0) then
          put "Successfully opened WORK.ONE in"
              "UTILITY mode.";
     end;

   put "Create a check integrity constraint named teen.";
   rc = iccreate(dsid, 'teen', 'check',
   '(age > 12) && (age < 20)');

   if (rc > 0) then
      do;
       put rc=;
       _msg_=sysmsg();
       put _msg_=;
     end;
     else do;
      put "Successfully created a check"
          "integrity constraint.";
     end;
```

```
 put "Create a not-null integrity constraint named nn.";
 rc = iccreate(dsid, 'nn', 'not-null', 'age');

 if (rc > 0) then
    do;
     put rc=;
     _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
    put "Successfully created a not-null"
        "integrity constraint.";
   end;

 put "Create a unique integrity constraint named uq.";
 rc = iccreate(dsid, 'uq', 'unique', 'age');

 if (rc > 0) then
    do;
     put rc=;
     _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
    put "Successfully created a unique"
        "integrity constraint.";
   end;

put "Create a primary key integrity constraint named pk.";
rc = iccreate(dsid, 'pk', 'Primary', 'name');

 if (rc > 0) then
    do;
     put rc=;
     _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
    put "Successfully created a primary key"
        "integrity constraint.";
   end;

 put "Closing WORK.ONE.";
 rc = close(dsid);
 if (rc > 0) then
   do;
    put rc=;
    _msg_=sysmsg();
    put _msg_=;
   end;

 put "Opening WORK.TWO in utility mode.";
 dsid2 = open('work.two', 'V');
    /*Utility mode */
 if (dsid2 = 0) then
```

```
     do;
    _msg_=sysmsg();
      put _msg_=;
     end;
     else do;
      if (dsid2 > 0) then
        put "Successfully opened WORK.TWO in"
            "UTILITY mode.";
     end;

   put "Create a foreign key integrity constraint named fk.";
   rc = iccreate(dsid2, 'fk', 'foreign', 'name',
   'work.one','null', 'restrict');

   if (rc > 0) then
      do;
       put rc=;
       _msg_=sysmsg();
       put _msg_=;
     end;
     else do;
      put "Successfully created a foreign key"
          "integrity constraint.";
     end;

 put "Closing WORK.TWO.";
   rc = close(dsid2);
   if (rc > 0) then
     do;
      put rc=;
      _msg_=sysmsg();
      put _msg_=;
     end;
return;

TERM:
  put "End of test SCL integrity constraint"
      "functions.";
return;
```

上記のプログラムにより、SCL カタログエントリが作成されます。次のプログラムでは、One および Two という名前の 2 つのデータファイルを作成し、SCL エントリ Example.Ic_Cat_Allics.SCL を実行します。

```
      /* Submit to create data files. */

   data one two;
      input name $ age;
   datalines;
   Morris 13
   Elaine 14
   Tina 15
   ;

      /* after compiling, run the SCL program */

   proc display catalog= example.ic_cat.allics.scl;
```

```
   run;
```

一貫性制約の削除

次の例では、一貫性制約を削除します。主キー一貫性制約を削除する場合は、先に外部キー一貫性制約を削除しておく必要があります。

次のプログラムセグメントでは、PROC SQL を使用して一貫性制約を削除します。

```
proc sql;
   alter table salary
     DROP CONSTRAINT for_key;
   alter table people
       DROP CONSTRAINT gender
       DROP CONSTRAINT _nm0001_
       DROP CONSTRAINT status
       DROP CONSTRAINT prim_key
    ;
quit;
```

次のプログラムセグメントでは、PROC DATASETS を使用して一貫性制約を削除します。

```
proc datasets nolist;
   modify tv_survey;
      ic delete val_max;
      ic delete val_gender;
      ic delete val_age;
run;
quit;
```

次のプログラムセグメントでは、SCL を使用して一貫性制約を削除します。

```
TERM:
   put "Opening WORK.TWO in utility mode.";
   dsid2 = open( 'work.two' , 'V' );  /* Utility mode.     */
   if (dsid2 = 0) then
     do;
     _msg_=sysmsg();
     put _msg_=;
   end;
   else do;
     if (dsid2 > 0) then
        put "Successfully opened WORK.TWO in Utility mode.";
    end;

rc = icdelete(dsid2, 'fk');
if (rc > 0) then
  do;
   put rc=;
   _msg_=sysmsg();
  end;
else
   do;
   put "Successfully deleted a foreign key integrity constraint.";
   end;
rc = close(dsid2);
return;
```

非アクティブ一貫性制約の再アクティブ化

次のプログラムセグメントでは、COPY、CPORT、CIMPORT、UPLOAD、DOWNLOAD プロシジャの実行により非アクティブ状態になっていた外部キー一貫性制約を、再びアクティブにします。

```
proc datasets;
   modify SAS-data-set;
      ic reactivate fkname references
libref;
   run;
quit;
```

主キー制約と外部キー制約の重複の定義

次のコードでは、重複する主キー制約と外部キー制約の定義を示しています。

```
   data Singers1;
      input FirstName $ LastName $ Age;
      datalines;
   Tom Jones 62
   Kris Kristofferson 66
   Willie Nelson 69
   Barbra Streisand 60
   Paul McCartney 60
   Randy Travis 43
   ;
   data Singers2;
      input FirstName $ LastName $ Style $;
      datalines;
   Tom Jones Rock
   Kris Kristofferson Country
   Willie Nelson Country
   Barbra Streisand Contemporary
   Paul McCartney Rock
   Randy Travis Country
   ;
   proc datasets library=work nolist;
      modify Singers1;
         ic create primary key (FirstName LastName); 1
      run;
      modify Singers2;
         ic create foreign key (FirstName LastName) references Singers1
            on delete restrict on update restrict;  2
      run;
      modify Singers2;
         ic create primary key (LastName FirstName);  3
      run;
      modify Singers1;
         ic create foreign key (LastName FirstName) references Singers2
            on delete restrict on update restrict;  4
      run;

   quit;
```

1 データセット Singers1 の変数 FirstName および LastName に関する主キー制約を定義します。

2 データセット Singers2 の変数 FirstName および LastName に関する外部キー制約を定義します。この外部キーは、ステップ 1 で定義した主キーを参照します。同じ変数を使用して主キーを定義することが目的であるため、外部キーの更新および削除に対して定義される参照アクションは、どちらも RESTRICT でなければなりません。

3 データセット Singers2 の変数 LastName および FirstName に関する主キー制約を定義します。まったく同じ変数がすでに外部キーとして定義されているため、これら 2 つの変数の順番を変える必要があります。

4 データセット Singers1 の変数 LastName および FirstName に関する外部キー制約を定義します。この外部キーは、ステップ 3 で定義した主キーを参照します。まったく同じ変数がすでに主キーとして定義されているため、これら 2 つの変数の順番を変える必要があります。これらの変数を使って主キーがすでに定義されているため、この外部キーの更新および削除に対して定義される参照アクションは、どちらも RESTRICT でなければなりません。

SAS インデックスについて

SAS インデックスの定義

インデックスとは、SAS データファイル用に作成するオプション指定のファイルです。インデックスを使用すると、特定のオブザベーションに直接アクセスできます。インデックスは、特定の変数の値を昇順で格納するほか、データファイルのオブザベーション内の値の場所に関する情報を含んでいます。言い換えれば、インデックスを使うと、値を基準としてオブザベーションを検索することが可能となります。

たとえば、123-45-6789 という SSN (社会保障番号)を含むオブザベーションを検索する場合を考えます。

- インデックスがない場合、SAS System は、データファイル内部の格納順にオブザベーションにシーケンシャルアクセスします。SAS System は各オブザベーションを読み込み、すべてのオブザベーションを読み込むまで、変数 SSN の値が 123–45–6789 であるオブザベーションを検索します。
- 変数 SSN にインデックスがある場合、SAS System は、オブザベーションに直接アクセスします。各オブザベーションを読み込まずに、インデックスを使用する条件を満たす値を含むオブザベーションに直接アクセスします。

データファイルの作成時にインデックスを作成するか、または既存のデータファイルに対してインデックスを作成することができます。データファイルは、圧縮または解凍することができます。データファイルごとに、1 つまたは複数のインデックスを作成できます。インデックスが存在する場合、SAS System は、インデックスをデータファイルの一部として処理します。つまり、オブザベーションを追加または削除したり、値を修正したりすると、インデックスは自動的に更新されます。

インデックスの利点

一般に、SAS System でインデックスを使用すると、次の状況においてパフォーマンスを向上させることができます。

- WHERE 式の処理でインデックスを使用すると、データのサブセットに、より高速かつより効率的にアクセスできます。WHERE 式を処理する場合、SAS System は、デフォルトでインデックスを使用する(ダイレクトアクセス)か、それともデータファイルを順番に読み取る(シーケンシャルアクセス)かを決定します。

- BY グループ処理でインデックスを使用すると、オブザベーションがインデックス順(値の昇順)に返されます。データファイルがインデックス順に格納されていない場合でも、SORT プロシジャを使用する必要はありません。

  注: SORT プロシジャを使用すると、インデックスは使用されません。

- SET ステートメントと MODIFY ステートメントの場合、KEY=オプションを使用することによって、DATA ステップでインデックスを指定して、データファイルの特定のオブザベーションを取得することができます。

また、インデックスの利点は、SAS System の別の部分でも得られます。SCL (SAS コンポーネント言語)でインデックスを使用すると、テーブル探索操作のパフォーマンスが向上します。SQL プロシジャでインデックスを使用すると、結合クエリなど、特定のクエリをより効率的に処理できます。SAS/IML ソフトウェアでは、読み取り、削除、一覧表示、追加の操作にインデックスを使用することを明示的に指定できます。

インデックスを使用すると、特に、大きなデータファイルの場合に、オブザベーションを特定する時間を短縮できますが、インデックスの作成、格納、管理を行うためのリソース使用量は増加します。インデックスを作成するかどうかを決定する場合、パフォーマンスの向上だけでなく、リソース使用量の増加も考慮する必要があります。

注: DATA ステップでのサブセット化 IF ステートメント、または FSEDIT プロシジャでの FIND コマンドおよび SEARCH コマンドでは、インデックスは使用できません。

インデックスファイル

インデックスファイルとは、関連付けられているデータファイルと同一の名前でメンバタイプ INDEX を持つ SAS ファイルです。インデックスファイルはデータファイルごとに 1 つだけ存在します。これは、1 つのデータファイルのすべてのインデックスが単一のファイル内に格納されることを意味します。

インデックスファイルは、操作環境に応じて、別ファイルである場合とデータファイルの一部である場合とがあります。どちらの場合も、インデックスファイルは、データファイルと同一の SAS ライブラリに格納されます。

インデックスファイルはエントリで構成されます。エントリは階層的に編成され、ポインタで接続されます。これらのエントリは、SAS System によって管理されます。インデックスファイル階層の最下位レベルは、インデックス付き変数の個別値を昇順に表すエントリとして構成されます。各エントリには次の情報が含まれます。

- 個別値
- 各オブザベーションを識別する 1 つまたは複数の一意のレコード識別子(これを RID と呼びます)。RID は内部オブザベーション番号と見なすことができます。

インデックスファイルでは、各値の後ろに 1 つまたは複数の RID が続きます。この RID は、値を含むデータファイルのオブザベーションを識別します。(複数の RID が生成されるのは、同一値が複数現れた場合です)。たとえば、次の例は、変数 LastName のインデックスファイルエントリを示します。

表 26.8 *インデックスファイルエントリ*

値	レコード識別子
Avery	10
Brown	6, 22, 43
Craig	5, 50

値	レコード識別子
Dunn	1

インデックスを使用して WHERE 式などを処理する場合、SAS System は、インデックスファイルをバイナリ検索し、指定された値を含む最初のエントリにインデックスを配置します。次に、その値の RID を使用して、その値を含むオブザベーションを読み込みます。1 つの値が複数の RID を持つ場合(例: 前述した例の Brown の値)、SAS System は、リスト内の次の RID が指しているオブザベーションを読み込みます。その結果、値または値の範囲で指定されているオブザベーションをすばやく特定できます。

たとえば、インデックスを使用して次の WHERE 式 **`where age > 20 and age < 35;`**を処理する場合、SAS System は、インデックスで最初の値(20 よりも大きい)のインデックスエントリを指し、その値の RID を使用してオブザベーションを読み込みます。次に、35 以上の値のインデックスエントリが見つかるまで、オブザベーションを読み込みながらインデックスエントリを順番に移動します。

SAS System は、更新のたびに自動的にインデックスファイルをバランスのとれた一定の状態に保ちます。つまり、任意のインデックスエントリにアクセスするためのリソース効率が一定になるようにします。さらに、削除された値の占有領域をすべて回復して再利用します。

インデックスの種類

単一インデックスと複合インデックス

インデックスを作成する場合は、インデックスを付ける変数を指定します。インデックス付き変数のことを、キー変数と呼びます。作成できるインデックスは、次の 2 種類です。

- 単一インデックス(1 つのキー変数の値で構成される場合)
- 複合インデックス(複数のキー変数の値で構成される場合)

単一インデックスまたは複合インデックスを作成することだけでなく、インデックス(およびそのデータファイル)を一意の値に制限することや、インデックスから欠損値を除外することもできます。

単一インデックス

最も一般的なインデックスが単一インデックスです。これは、1 つのキー変数の値を含むインデックスです。キー変数の種類は、数値でも文字でもかまいません。単一インデックスを作成すると、キー変数の名前がインデックスの名前にも使用されます。

次の例では、DATASETS プロシジャステートメントを使用して、データファイル College.Survey の変数 Class および Major に対する 2 つの単一インデックスを作成します。

```
proc datasets library=college;
   modify survey;
      index create class;
      index create major;
run;
```

インデックスを使用して WHERE 式を処理するために、SAS System はインデックスを 1 つだけ使用します。WHERE 式に複数のキー変数を使用する条件が含まれている場合、最も小さなサブセットを指定する条件を特定します。たとえば、データセット College.Survey には次のデータが含まれているとします。

- 42,000 個のオブザベーションに **class=12** が含まれています。
- 6,000 個のオブザベーションに **major='Biology'** が含まれています。
- 350 個のオブザベーションに **class=12** と **major='Biology'** の両方が含まれています。

変数 Class および Major の単一インデックスの場合、Major を選択して次の WHERE 式を処理します。

```
where class=12 and major='Biology';
```

複合インデックス

複合インデックスとは、複数のキー変数の値を含むインデックスです。それらの変数の値は単一の値を形成するように連結されます。キー変数には、数値変数、文字変数、またはその組み合わせを指定できます。例としては、変数 LastName および FirstName の複合インデックスが挙げられます。このインデックスの値は、同一のオブザベーションにある LastName の値とそれに続く FirstName の値とで構成されます。複合インデックスを作成する場合は、一意のインデックス名を指定する必要があります。

次の例では、DATASETS プロシジャステートメントを使用して、2 つのキー変数 ZipCode および SchoolId を指定して、データセット College.MailList の複合インデックスを作成します。

```
proc datasets library=college;
   modify maillist;
      index create zipid=(zipcode schoolid);
run;
```

複合インデックスの最初の変数だけを使用することがあります。たとえば、ZipCode および SchoolId の複合インデックスの場合、次の WHERE 式では、変数 ZipCode が複合インデックスの最初のキー変数であるため、複合インデックスを変数 ZipCode に使用できます。

```
where zipcode = 78753;
```

ただし、複合インデックスのキー変数をすべて使用するように WHERE 式を記述した方が、処理が速くなります。この方法を複合最適化と呼びます。複合最適化は、複合インデックスを使用して、複数の WHERE 式の条件を最適化するプロセスです。次の WHERE 式を使用すると、複合インデックスを使用して、ZIPCODE='78753'および SCHOOLID='55'を持つオブザベーションがすべて検索されます。このように記述することにより、すべての条件がインデックスの単一の検索で処理されます。

```
where zipcode = 78753 and schoolid = 55;
```

単一インデックスと複合インデックスのどちらを作成するかを決定する場合は、データへのアクセス方法を考慮します。単一の変数のデータに頻繁にアクセスする場合は、単一インデックスが適しています。一方、複数の変数のデータに頻繁にアクセスする場合は、複合インデックスが適しています。

重複しない値

社会保障番号や従業員番号のように、変数の値が一意であることが重要である場合がよくあります。キー変数に対して一意な値を宣言するには、UNIQUE オプションを指定してインデックスを作成します。UNIQUE オプションを指定すると、キー変数とオブザベーションが 1 対 1 に対応するようなインデックスを作成できます。また、キー変数に重複した値を持つオブザベーションを追加することは拒否されます。

次の例は、変数 IdNum の単一インデックスを作成し、IdNum のすべての値が重複しないことを指定します。

```
proc datasets library=college;
```

```
   modify student;
      index create idnum / unique;
run;
```

欠損値

変数に多くの欠損値がある場合、欠損値がインデックスに含まれないように指定する必要があります。インデックスを作成する場合は、NOMISS オプションを使用して、インデックスが欠損値を保持しないように指定します。

次の例は、変数 Religion の単一インデックスを作成し、インデックスが変数の欠損値を保持しないように指定します。

```
proc datasets library=college;
   modify student;
      index create religion / nomiss;
run;
```

UNIQUE オプションとは対照的に、キー変数の欠損値を含むオブザベーションをデータファイルに追加することができます。ただし、欠損値はインデックスに追加されません。

BY グループ処理や、欠損値を含むオブザベーションを指定する WHERE 式の処理には、NOMISS オプションを使用して作成したインデックスは使用されません。欠損値が存在しない場合、SAS System は、BY ステートメントや WHERE 式の処理でインデックスを使用することを検討します。

次の例では、インデックス Age が NOMISS オプションを使用して作成され、変数 Age の欠損値を含むオブザベーションが存在するとします。この場合、SAS System はインデックスを使用しません。

```
proc print data=mydata.employee;
   where age < 35;
run;
```

インデックス作成のための注意点

インデックスのコスト

インデックスを作成することにより、パフォーマンスは向上します。しかし、インデックスは、一部のリソースを保護する一方で、その他のリソースを消費します。したがって、インデックスの作成、使用、管理に関連するリソース効率について考慮する必要があります。本セクションでは、リソース使用量に関する情報や、インデックスを作成するためのガイドラインを示します。

CPU コスト

インデックスを作成したり、データファイルの変更時にインデックスを保持したりするには、より多くの CPU 時間が必要になります。つまり、インデックス付きデータファイルの場合、値が追加、削除、修正されると、該当するインデックスの値も追加、削除、修正されます。

インデックスを使用してデータファイルからオブザベーションを読み取る場合も、CPU 使用率が増加します。CPU 使用率が増加するのは、データを順番に読み取る場合よりも複雑な処理が行われるためです。CPU 使用率は増加しますが、条件を満たすオブザベーションのみを直接読み取ることができます。つまり、条件を満たすオブザベーションが多数存在する場合は、インデックスを使用すると CPU 使用率が増加します。

注: 一部の動作環境では、インデックスを使用する場合と使用しない場合の CPU 使用率を比較するために、STIMER または FULLSTIMER システムオプションを使用して、パフォーマンス統計量を SAS ログに表示することができます。

I/O コスト

インデックスを使用してデータファイルからオブザベーションを読み取ると、データファイルを順番に読み取る場合と比較して、I/O(入出力)要求の回数が増加する場合があります。たとえば、インデックスを使用して BY ステートメントを処理すると、入出力回数が増加する場合があります。ただし、SORT プロシジャの実行を省略することができます。WHERE 式による処理の場合、SAS System は、インデックスを使用するかどうかを決定するのに入出力回数を考慮します。

1. インデックスファイルがバイナリ検索され、指定された値を含む最初のエントリにインデックスが配置されます。
2. SAS System は、値の RID (識別子)を使用して、指定された値を含むオブザベーションに直接アクセスします。オブザベーションが外部記憶装置からバッファに転送されます。バッファとは、データの読み書きを行うために一時的に利用するメモリ領域です。データはページ単位で転送されます。ページとは、1 回の入出力要求で転送できるデータ量(オブザベーション数)です。データファイルのページサイズは、BUFSIZE データセットオプションで指定することができます。
3. WHERE 式の条件を満たすまで処理を続行します。SAS System がオブザベーションにアクセスするたびに、オブザベーションを含むデータファイルのページをメモリに読み込みます。ただし、すでにメモリに存在している場合は読み込みません。つまり、オブザベーションがデータファイルの複数のページに存在する場合は、オブザベーションごとに入出力動作を実行します。

結果として、データがランダムに分布しているほど、インデックスを使用するための入出力要求の回数が増加します。データがインデックスのように整列しているほど、データにアクセスするための入出力要求の回数が減少します。

指定したバッファ数によって、同時にメモリに存在できるページ数が決定します。多くの場合、バッファ数が増加するほど、必要となる入出力の回数が減少します。ページサイズが 4096 バイトで、1 つのバッファが割り当てられている場合は、1 回の入出力で 4096 バイトのデータ(1 ページ)が転送できます。入出力回数を減少させるには、ページサイズ(バッファサイズ)を増やします。ただし、メモリ上により大きなバッファが必要となります。ページサイズを減少させると、入出力回数は増加します。

データファイルのページサイズやページ数など、データファイル特性の情報を取得するには、CONTENTS プロシジャを実行するか、DATASETS プロシジャで CONTENTS ステートメントを使用します。この情報を使用して、データファイルのページサイズを確認したり、さまざまなページサイズをテストすることができます。CONTENTS プロシジャで取得できる情報は、動作環境によって異なります。

BUFSIZE=データセットオプション(またはシステムオプション)は、データファイルの作成時に、データファイルの永久ページサイズ(バッファサイズ)を設定します。ページサイズとは、1 回の入出力操作で 1 つのバッファに転送可能なデータ量のことです。BUFNO=データセットオプション(またはシステムオプション)は、1 つのデータファイル向けに割り当てるバッファ数、または SAS プログラムを実行するシステム全体に割り当てるバッファ数を指定します。すなわち、BUFNO=オプションはデータセット属性としては保存されません。

バッファの必要条件

SAS System は、インデックスの作成および管理に使用するリソースに加えて、インデックスを実際に使用する際にバッファ用の追加的なメモリを必要とします。データファイルを開くと、インデックスファイルを開きますが、インデックスは開きません。インデック

スを使用するときまでバッファは使用されませんが、インデックスが使用される場合に備えてバッファが割り当てられている必要があります。

割り当てられるバッファ数は、インデックスの階層の数と、データファイルを開くモードによって異なります。データファイルを入力用に開く場合のバッファの最大数は 3 で、更新用に開く場合の最大数は 4 です。(これらのバッファは別の用途にも利用できます。インデックス専用ではありません)。

IBUFSIZE=システムオプションは、インデックスファイルの作成時に、同インデックスファイルのディスク上のページサイズを指定します。デフォルト設定では、SAS System は、動作環境に最適な最小ページサイズを使用します。通常、インデックスのページサイズを指定する必要はありません。ただし、状況によっては、デフォルト以外のページサイズが必要となる場合もあります。詳細については、“IBUFSIZE= System Option” (*SAS System Options: Reference*)を参照してください。

IBUFNO=システムオプションは、インデックスファイルのナビゲート時に割り当てられる拡張バッファの数を指定します。デフォルトでは、SAS System は自動的に最小数のバッファを割り当てます。通常、拡張バッファを指定する必要はありません。ただし、IBUFNO=システムオプションを使用すると、特定のインデックスファイルに必要となる入出力操作の回数を制限することにより、実行時間を短縮できる場合があります。なお、この場合、実行時間は短縮されますが、その代償としてメモリの消費量が増えることになります。詳細については、“IBUFNO= System Option” (*SAS System Options: Reference*)を参照してください。

ディスク領域の必要条件

インデックスファイルを格納するためには、追加のディスク領域が必要になります。インデックスファイルは、動作環境に応じて、別ファイルである場合とデータファイルの一部である場合とがあります。

インデックスファイルのサイズ情報を取得するには、CONTENTS プロシジャ、または DATASETS プロシジャで CONTENTS ステートメントを使用します。CONTENTS プロシジャで取得できる情報は、動作環境によって異なります。

インデックスの作成ガイドライン

データファイルの留意点

- 小さなデータファイルの場合、シーケンシャルな処理とインデックスによるダイレクトな処理の効率が同等になることがよくあります。データファイルのページ数が 3 ページ未満の場合は、インデックスの作成は適していません。データにシーケンシャルアクセスするほうが高速になります。データファイルのページ数を確認するには、CONTENTS プロシジャを使用するか、DATASETS プロシジャの CONTENTS ステートメントを使用します。CONTENTS プロシジャで取得できる情報は、動作環境によって異なります。
- 頻繁に変更されるデータファイルの場合は、インデックスのリソース効率を考慮します。頻繁に変更されるファイルでは、変更ごとのインデックス更新に関連するオーバーヘッドによって、インデックスを使用したデータアクセスの処理効率が減少する可能性があります。
- 大きなデータファイルから小さなオブザベーションのサブセットを取得する場合(たとえば、オブザベーション全体の 25%未満を取得する場合)は、インデックスを作成します。この場合、データファイルのページを処理するリソース効率は、データファイル全体を順番に読み取るときのオーバーヘッドよりも少なくなります。サブセットが小さいほど、パフォーマンスが向上します。
- インデックス作成時の入出力回数を減少させるには、最初にデータをキー変数順に並べ替えられます。その後、パフォーマンスを向上させるために、キー変数で並

べ替えられた順序でデータファイルを保持します。この方法により、同じ値がグループ化されるため、入出力回数が減少します。つまり、データファイルがキー変数順に整列しているほど、インデックスを効率良く使用できます。データファイルが複数のインデックスを持つ場合は、最も頻繁に使用するキー変数順にデータを並べ替えます。

- 条件を満たすためにデータが適切に並べ替えられているならば、インデックスを使用した WHERE 式の最適化が必要ない場合もあります。インデックスを使わずに WHERE 式を処理する場合、SAS System はまず、以前 SORT プロシジャによりファイルに格納されたソートインジケータをチェックします。ソートインジケータが適切である場合、SAS System は、その WHERE 式を満たす値が存在しなくなった時点でファイルの読み取りを停止します。たとえば、ファイルがインデックスを使用せずに、年齢を表す変数 Age で並べ替えられている場合を考えてみましょう。式 **`where age le 25`** を処理する場合、SAS は、25 を超えるオブザベーションが見つかると、オブザベーションの読み込みを停止します。SAS はオブザベーションの読み込みを停止するタイミングを判定できますが、インデックスが存在しない場合、どこから開始するかを示す情報がありません。インデックスがない場合、SAS System は常に最初のオブザベーションから処理を開始します。この場合、大量のオブザベーションの読み込みが必要となることがあります。

インデックスの使用の留意点

- ディスク容量や更新時のリソース使用量などを節約するには、データファイル 1 つあたりのインデックス数を最小限に抑えます。
- 処理におけるインデックスの使用頻度を考慮します。インデックスを頻繁に使用すると、インデックスの作成および管理に使用されるリソースが増加します。つまり、WHERE 式による直接処理よりも、効率が減少する可能性があります。インデックスを実際に作成したり、データファイルが変更されたりするたびに、インデックスを管理するのに必要なリソースを見極めて、インデックスを作成するかどうかを決定します。
- インデックスを作成して WHERE 式を処理する場合は、1 つのインデックスですべて条件式を満たすようなインデックスを作成すべきではありません。条件式に複数の変数を記述する場合は、一意的に値を識別できる変数に対して単一インデックスを使用すると、その条件式を満たす可能性が高くなります。

キー変数候補

ほとんどの場合、データファイルに対するクエリには、複数の変数を使用します。ただし、特定の変数がその他の変数よりもキー変数として適している場合は、データファイルのすべての変数にインデックスを付けることは避けます。

- キー変数としてインデックスを付ける変数は、条件式で使用される変数である必要があります。つまり、条件式に指定される変数をキー変数とすることにより、条件式に該当するオブザベーションを検索してサブセット化するのに、効率的にインデックスを使用します。加えて、キー変数は、他の変数と一緒に条件式に使用される変数である必要があります。
- ある変数を使用して、WHERE 式によりオブザベーションを特定できる場合、指定した変数はインデックスを付けるキー変数に適しています。つまり、キー変数は、一意的に識別できる値である必要があります。これは、インデックスを使用することにより、特定のオブザベーションを最小限で選択できることを意味します。たとえば、Age (年齢)、FirstName (名前)、Gender (性別)などの変数は、一意的に識別できない値です。これは、データ表現の大部分に、同一の年齢、名前、性別が含まれる可能性が高いためです。他方、LastName (姓)などの変数は適切といえます。これは、多くの従業員が同じ姓を持つ可能性が少ないためです。

  たとえば、変数 LastName と Gender があるデータファイルを考えます。

- データファイルへのクエリの多くが変数 LastName を含む場合は、LastName にインデックスを付けると便利です。この変数の値は通常、一意的に識別できるからです。ただし、大多数のクエリが変数 Gender を含む場合、インデックスを付けるには適しません。変数 Gender は、値の半分が男性、残りの半分が女性というように一意的に識別できる値ではないためです。
- ただし、データファイルへのクエリのほとんどが、次の WHERE 式のように、変数 LastName と Gender の両方を含む場合は、LastName と Gender の複合インデックスを作成すると、パフォーマンスが向上する可能性があります。

```
where lastname='LeVoux' and gender='F';
```

複合インデックスを作成する場合は、一意的に識別できる変数を最初のキー変数にする必要があります。

インデックスの作成

インデックス作成の概要

データファイルのインデックスを 1 つ作成する場合、単一インデックスまたは複合インデックスのどちらかを選択できます。インデックスを複数作成する場合は、複数の単一インデックス、複数の複合インデックス、または単一インデックスと複合インデックスの組み合わせを選択できます。

1. DATASETS プロシジャの INDEX CREATE ステートメントなどの方法を使用して、1 つまたは複数の変数に対するインデックスを作成します。
2. データファイルを一度に 1 オブザベーションずつ読み込み、キー変数ごとに値と RID を抽出してインデックスファイルに配置します。

SAS System は、インデックス内に配置されている値が連続して同じであるか、または増えていることを確認します。SAS System は、データがすでにキー変数によって昇順に並べ替えられているかどうかを判定します。この判定は、データファイル内のソートインジケータを検査することにより行われます。ソートインジケータとは、データの並べ替え状態を示すファイル属性のことです。ソートインジケータは、SAS データファイルのディスクリプタ情報とともに格納されており、以前の SORT プロシジャまたは SORTEDBY=データセットオプションにより設定されます。

ソートインジケータの値が昇順になっている場合、インデックスファイルの値を並べ替える必要がないため、リソース使用量を節約できます。SAS System は常に、データが指示どおりに並べ替えられていることを検証します。データが指示どおりに並べ替えられていない場合、インデックスは作成されません。たとえば、SORTEDBY=データセットオプションによりソートインジケータは設定されたが、データが指示どおりに並べ替えられていない場合、エラーが発生します。値が昇順に並べ替えられていないためにインデックスが作成されなかったことを伝えるメッセージが SAS ログに書き出されます。

ソートインジケータ内の値が昇順に並べ替えられていない場合、SAS System は、インデックスに含まれているデータを昇順に並べ替えます。データを並べ替える場合、SAS System は次の手順に従います。

1. SAS System は、まずスレッド対応の並べ替えを使用してデータの並べ替えを試みます。並べ替え処理を独立した複数の実行可能プロセスに分割することにより、データの並べ替えにかかる時間を短縮できます。スレッド対応の並べ替えを使用するためには、インデックスのサイズが十分大きいことが必要です(これは SAS により判定されます)。また、CPUCOUNT=システムオプションにより複数のプロセッサを使用するよう設定されていること、および THREADS システムオプションが有効になっていることも必要です。さらに、スレッド対応の並べ替えでは、十分な量のメモリが利用できることが必要となります。十分な量のメモリが利用できない場合、

SAS System はスレッドの数を減らした後、並べ替え処理を再開します。この結果、インデックスの作成にかかる時間が長くなります。

2. スレッド対応の並べ替えを実行できない場合、SAS System はスレッド化されていない並べ替えを使用します。

注: 使用される並べ替えの種類、メモリおよびリソース情報、作成されるインデックスの状態に関するメッセージを表示するには、次に示すように、SAS システムオプション MSGLEVEL=に値 I を設定します。

```
options msglevel=i;
```

DATASETS プロシジャの使用

DATASETS プロシジャでは、インデックスを作成および削除するためのステートメントを使用できます。次の例では、MODIFY ステートメントはデータファイルを識別し、INDEX DELETE ステートメントは 2 つのインデックスを削除します。また、2 つの INDEX CREATE ステートメントは、インデックスを付ける変数を指定し、最初の INDEX CREATE ステートメントでは、UNIQUE オプションおよび NOMISS オプションを指定します。

```
proc datasets library=mylib;
modify employee;
   index delete salary age;
   index create empnum / unique nomiss;
   index create names=(lastname firstname);
```

注: 同じ DATA ステップでインデックスの削除と作成を行う場合は、INDEX CREATE ステートメントの前に INDEX DELETE ステートメントを配置します。このようにすると、削除されたインデックスの占有領域を、インデックス作成時に再利用できます。

INDEX=データセットオプションの使用

データファイルの作成時に DATA ステップでインデックスを作成するには、INDEX=データセットオプションを使用します。INDEX=データセットオプションには、NOMISS オプションおよび UNIQUE オプションを指定することもできます。次の例では、変数 Stock の単一インデックスを作成し、UNIQUE オプションを指定します。

```
data finances(index=(stock /unique));
```

次の例では、変数 SSN、City、State を使用して、SSN という名前の単一インデックスと、CitySt という名前の複合インデックスを作成します。

```
data employee(index=(ssn cityst=(city state)));
```

SQL プロシジャの使用

SQL プロシジャは、インデックスの作成と削除、UNIQUE オプションをサポートします。変数リストでは、変数名を、ブランク(SAS 規則)ではなくカンマ(SQL 規則)で区切る必要があります。

DROP INDEX ステートメントは、インデックスを削除します。CREATE INDEX ステートメントは、UNIQUE オプション、インデックス名、対象とするデータファイル、インデックスを付ける変数を指定します。次にその例を示します。

```
drop index salary from employee;
create unique index empnum on employee (empnum);
create index names on employee (lastname, firstname);
```

その他の SAS プロダクトの使用

SAS/CONNECT ソフトウェア、SAS/IML ソフトウェア、SAS Component Language、SAS/Warehouse Administrator などの、その他の SAS Utilities やプロダクトを使用することでも、インデックスの作成や削除が行えます。

WHERE 処理でのインデックスの使用

WHERE 処理でのインデックスの使用の概要

WHERE 処理は、WHERE 式を実行したときに、処理するオブザベーションを条件に従って選択します。インデックスを使用して WHERE 式を処理すると、パフォーマンスが向上します。この方法を WHERE 式の最適化と呼びます。

WHERE 式を処理するにあたって、SAS System はデフォルトで、インデックスを使用するか、データファイルのすべてのオブザベーションを順番に読み取るかを決定します。この決定を行うために、SAS System は次の処理を行います。

1. 利用可能なインデックスを識別します。
2. 扱うオブザベーション数を推定します。複数のインデックスを使用できる場合、SAS System は、オブザベーションが最小サブセットとなる(選択するオブザベーション数が最も少ない)インデックスを選択します。
3. リソース使用量を比較し、WHERE 式を満たすのに、インデックスを使用するダイレクトアクセスが効率的か、すべてのオブザベーションを順番に読み取るシーケンスアクセスが効率的かを決定します。

   注: SAS System は、インデックスを使用するかどうかを決定する場合、複数の要因を検討します。このため、最適なパフォーマンスを確保するためには、何よりも経験を積むことが必要です。繰り返し使用する WHERE 式が存在する場合、インデックスを使用した結果とインデックスなしの結果とを比較することにより、どちらの方法が最適なパフォーマンスを提供するかを判定できます。インデックスの使用を制御するには、IDXWHERE=および IDXNAME=データセットオプションを使用します。“データセットオプションを使用して WHERE 処理でのインデックスの使用を制御する” (647 ページ)を参照してください。

使用可能なインデックスの識別

WHERE 式を処理するためにインデックスを使用するかどうかを SAS System が決定する最初のステップでは、WHERE 式に含まれる変数がキー変数かどうか、インデックスを持っているかどうかの識別が行われます。複数の条件で異なる変数を指定する WHERE 式を記述している場合でも、SAS System は、インデックスを 1 つだけ使用して WHERE 式を処理します。SAS System は、ほとんどの条件を満たし、かつオブザベーションの最小サブセットを選択するインデックスを選択します。

- 通常、SAS System は 1 つの条件を選択します。条件に指定された変数は、単一インデックスのキー変数か、複合インデックスの最初のキー変数になります。
- ただし、適切な WHERE 式を記述することによって、SAS System は複合インデックスの複数のキー変数を使用します。この方法を、複合最適化と呼びます。“複合最適化” (644 ページ)を参照してください。

SAS System は、次の WHERE 式の条件式に対してインデックスの使用を試みます。

表 26.9 最適化できる WHERE 条件

条件	複合化最適化で有効	例
比較演算子(EQ 演算子、大なりまたは小なりなどによる比較、IN 演算子など)	有効	where empnum eq 3374; where empnum < 2000; where state in ('NC','TX');
NOT を使用した比較演算子	有効	where empnum ^= 3374; where x not in (5,10);
コロン修飾子を使用した比較演算子	有効	where lastname gt:'Sm';
CONTAINS 演算子	無効	where lastname contains 'Sm';
上限と下限の両方による明確な範囲を指定した演算子(BETWEEN-AND 演算子など)	有効	where 1 < x < 10; where empnum between 500 and 1000;
パターンマッチ演算子、LIKE 演算子、NOT LIKE 演算子	無効	where firstname like '%Rob_%'
IS NULL 演算子または IS MISSING 演算子	無効	where name is null; where idnum is missing;
TRIM 関数	無効	where trim(state)='Texas';
次の形式での SUBSTR (left of =)関数: WHERE SUBSTR (*variable, position <,length>*)='*string*'; ただし、次の条件を満たす場合 *position* には開始文字位置を表す数値定数を指定し、その値は *variable* の長さ以下であること *length* には、*string* の長さを表す数値定数を指定すること *length* と *position* の合計が、*variable* の長さに 1 を加えた値以下であること	無効	where substr (month,4,5)='ember' and (city='Charleston' or city='Atlanta');

注: 算術演算子、条件式、SOUNDSLIKE (=*)演算子、前述した TRIM 関数と SUBSTR 関数以外のすべての関数では、インデックスを使用して条件を最適化できません。

次の例は、単一条件の最適化を示しています。

- 次の WHERE 式の場合は、変数 Major をキー変数として持つ単一インデックスを使用します。

```
where major in ('Biology', 'Chemistry', 'Agriculture');
where class=11 and major in ('Biology', 'Agriculture');
```

- すでに例で示した変数 ZipCode および SchoolId を持つ複合インデックスの場合、SAS System は複合インデックスを使用して、次の条件を満たすことができます。これは、変数 ZipCode が複合インデックスの最初のキー変数であるためです。

```
where zipcode = 78753;
```

ただし、次の条件式では複合インデックスを使用できません。これは、変数 SchoolId が複合インデックスの最初のキー変数ではないためです。

```
where schoolid gt 1000;
```

複合最適化

複合最適化は、1 つの複合インデックスを使用して、複数の WHERE 式の条件を最適化するプロセスです。複合インデックスを使用して条件を最適化すると、パフォーマンスが大幅に向上する可能性があります。

たとえば、変数 LastName と FirstName の複合インデックスがあるとします。次の WHERE 式を実行する場合、SAS System は、最初の 2 つの変数の値を連結して条件の最適化を行い、変数 EmpId に指定された値を満たすオブザベーションを取得します。

```
where lastname eq 'Smith' and firstname eq 'John' and empid=3374;
```

複合最適化が行われるようにするには、次の条件をすべて満たす必要があります。

- 複合インデックスにある少なくとも最初の 2 つのキー変数が有効な WHERE 式の条件で使用されていること。複合最適化で有効な条件の一覧については、表 26.9 (643 ページ)を参照してください。
- 少なくとも 1 つの条件が EQ または IN 演算子を使用していること。たとえば、すべての条件式を、明確な範囲を指定した演算子にすることはできません。
- 複数の条件式が AND または OR 論理演算子で接続されていること。
  - 複数の条件式が AND で接続されている場合、各条件式が並べられている順番には関係なく、条件式全体の真か偽かが決定されます。次に例を示します。

```
where lastname eq 'Smith'and firstname eq 'John';
```

  - 複数の条件式を OR で接続する場合、各条件式には同じ変数を指定すること。次に例を示します。

```
where firstname eq 'John' and
   (lastname eq 'Smith' or lastname eq 'Jones');
```

注: SAS System は、同じ変数を指定した複数の条件式を OR で接続した条件式を、IN 演算子を使用した単一の条件式へと変換します。たとえば、前述の WHERE 式の場合、SAS System は 2 つの条件式を OR で接続した条件式を、**`lastname IN ('Smith','Jones')`** へと変換します。その後、変数 FirstName と LastName の複合インデックスを使用して、Frstname が John で、かつ LastName が Smith か Jones のいずれかであるオブザベーションを選択します。

次の例では、変数 I、J、K に対して、IJK という名前の複合インデックスが存在するものとします。

- 次の WHERE 式条件は複合最適化されます。これは、各条件が複合インデックスの変数を指定し、サポートされる演算子を使用しているためです。SAS System は、

3 つの条件をすべて満たす最初のエントリに複合インデックスを配置し、3 つの条件すべてを満たすオブザベーションだけを取得します。

```
where I = 1 and J not in (3,4) and 'abc' < CH;
```

- 次の WHERE 式では、最初の 2 つの条件式が複合最適化されます。SAS System は、最初の 2 つの条件式を満たすオブザベーションのサブセットを取得すると、サブセットを調査し、3 つ目の条件式に一致しないオブザベーションをすべて除外します。

```
where I in (1,4) and J = 5 and K like '%c';
```

- 次の WHERE 式は、変数 I および J に関して複合最適化されます。SAS System は、2 つ目と 3 つ目の条件式を満たすオブザベーションを取得すると、サブセットを調査し、最初の条件式を満たさないオブザベーションを除外します。

```
where X < 5 and I = 1 and J = 2;
```

- 次の WHERE 式は、変数 I および J に関して複合最適化されます。

```
where X < Z and I = 1 and J = 2;
```

- 次の WHERE 式は複合最適化されません。これは、J および K が複合インデックスに含まれる最初の 2 つのキー変数ではないためです。

```
where J = 1 and K = 2;
```

- 次の WHERE 式は複合最適化されません。変数 I に関する比較条件が変数対変数であり、これはインデックス処理でサポートされていないためです。

```
where I < K and J in (3,4) and CH = 'abc';
```

少なくとも 1 つの条件が欠損値として評価されない限り、複合最適化を NOMISS 複合インデックスに関して実施できます。すなわち、複合最適化は、すべての条件の評価結果が欠損値となる場合、欠損値を保持できないインデックスである NOMISS インデックスに関しては実施できません。次の例は、変数 I、J、K に関する NOMISS 複合インデックスでの複合最適化を示しています。

- 次の WHERE 式は最適化されます。これは条件式 **K = 1** の評価結果が欠損値にはならないためです。

```
where I in (.,5) and J < 4 and K = 1;
```

- 次の WHERE 式は最適化されません。これは、各条件式の評価結果が欠損値となる可能性があるためです。

```
where I in (.,5) and J < 4 and K <= 1;
```

- 次の WHERE 式は最適化されません。これは、各条件式の評価結果が欠損値となる可能性があるためです。条件式 **J < 4** は、**J = .** の場合にオブザベーションを選択するため、これらのオブザベーションは NOMISS 複合インデックスでは表せません。

```
where I = . and J < 4 and .A < K < .D;
```

条件を満たすオブザベーション数の推定

SAS System は、WHERE 式を満たすインデックスを識別すると、使用可能なインデックスによって、条件を満たすオブザベーション数を推定します。複数のインデックスが存在する場合、SAS System は、条件を満たすオブザベーション数が最も少なくなるインデックスを選択します。

SAS System は、累積パーセント点(百分位数)と呼ばれる統計量を使用して、選択されるオブザベーションの数を推定します。百分位数情報はインデックスの値の分布を表すため、一様分布を前提とする必要はありません。インデックス付きデータファイルの百分位数情報を出力するには、CONTENTS プロシジャに CENTILES オプションを指

定するか、DATASETS プロシジャの CONTENTS ステートメントに CENTILES オプションを指定します(センタイル値として表示されます)。

デフォルトでは、データファイルが変更されるごとに百分位数情報が更新されることはありません。インデックスを作成するときに UPDATECENTILES オプションを含めると、百分位数情報の更新時期を指定できます。実際には、百分位数情報の更新は、データファイルを終了するごとに行う、特定の割合のキー変数値が変更されたときに行う、まったく行わない、のいずれかを指定することができます。また、DATASETS プロシジャで INDEX CENTILES ステートメントを指定すると、UPDATECENTILES オプションの値に関係なく、百分位数情報をすぐに更新することができます。

一般的な規則として、データファイルのオブザベーションのうち、全数の約 3 分の 1 以下が WHERE 式によって選択されると推定した場合に、インデックスを使用します。

注: パフォーマンスを向上させるために、SAS System が選択されるオブザベーションの数を推定する際に、次のことが行われます。

- 選択されるオブザベーション数がデータファイルの 3%未満である場合(または選択されるオブザベーションがない場合)、SAS System はインデックスを自動的に使用し、リソース使用量の比較は行いません。
- すべてのオブザベーションが選択される場合、デフォルトでは、IDXNAME=または IDXWHERE=データセットオプションが指定されない限り、SAS System はインデックスを使用しません。

リソースの使用量の比較

SAS System は、条件を満たすオブザベーション数を推定し、最も少ないオブザベーションを使用するインデックスを選択した後で、WHERE 式を満たすためには、インデックスを使用する方が高速である(リソース使用量が少ない)か、それともすべてのオブザベーションを順番に読み取る方が高速であるかを判定します。SAS System は、この決定を次のように行います。

- 選択されるオブザベーションがごく少数の場合は、データファイル全体を順番に検索するよりも、インデックスを使用する方が効率的です。
- オブザベーションの大半が選択される場合は、インデックスを使用するよりも、データファイルを順番に検索する方が効率的です。

これは、たとえば本の巻末にある索引を読者が使用するかどうかを決定するのと非常に似ています。本の索引は、読者が項目をページ番号で確認できるようになっています。索引を使用する場合、読者はそのページを開き、その項目だけを読みます。本に 42 の項目が含まれている場合、読者がごく少数の項目にだけ興味を持っているときは、索引を使用すると、他の項目を読まないで済むため、時間の節約になります。しかし、読者が 39 の項目に興味を持っている場合は、索引で各項目を検索すると、本全体を単純に読む場合よりも時間がかかります。

SAS System は、リソース使用量を比較するために、次の処理を行います。

1. インデックスを使用して WHERE 式を満たすのに必要な入出力の回数を予測します。このために、指定された値を含む最初のエントリにインデックスを配置します。現在利用できるバッファ数を考慮するバッファ管理シミュレーションで、インデックスページの RID (識別子)を処理し、データファイルのオブザベーションを読み取るのに必要な入出力の回数を示します。

   オブザベーションがデータファイルでランダムに分布している場合、SAS System はオブザベーションを確認するのに複数のデータファイルページを使用します。つまり、ページごとに入出力が必要になります。したがって、データがデータファイルでランダムに分布しているほど、インデックスを使用するのに必要な入出力の回数が増加します。データファイルのデータがインデックスのように整列しているほど、インデックスを使用するのに必要な入出力の回数が減少します。

2. データファイル全体を順番に検索する場合の入出力回数を計算し、2 つのリソース効率を比較します。

リソース効率の比較に影響する要因には、データファイルのサイズに対するサブセットのサイズ、データファイルの値の順序、データファイルのページサイズ、割り当てられたバッファ数、順番に読み取るための圧縮データファイルの解凍などがあります。

注: リソース効率の比較結果が等しい場合は、インデックスを選択します。

データセットオプションを使用して WHERE 処理でのインデックスの使用を制御する

WHERE 処理におけるインデックスの使用を制御するには、IDXWHERE=および IDXNAME=データセットオプションを使用します。

IDXWHERE=データセットオプションは、次のように、WHERE 式の条件を満たすインデックスを使用するかどうかについて、ソフトウェアの設定を上書きできます。

- IDXWHERE=YES を指定すると、データファイルを順番に検索する方がリソース効率がよくなる場合でも、最も適したインデックスを用いて WHERE 式を最適化します。
- IDXWHERE=NO を指定すると、SAS System はすべてのインデックスを無視し、データファイルを順番に検索して、WHERE 式の条件を満たします。
- IDXWHERE=データセットオプションを使用しても、インデックスを使用した BY ステートメントの処理が上書きされることはありません。

次の例は、WHERE 式の最適化に最も適したインデックスを使用することを指定します。SAS System は、データファイルを順番に検索する方がリソース効率がよくなる場合でも、それを無視します。

```
data mydata.empnew;
   set mydata.employee (idxwhere=yes);
   where empnum < 2000;
```

詳細については、*SAS データセットオプション: リファレンス*にある IDXWHERE データセットオプションの説明を参照してください。

IDXNAME=データセットオプションは、WHERE 式の条件を満たす特定のインデックスを使用するよう指定します。

IDXNAME=*index-name* のように、データファイルの単一インデックスまたは複合インデックスの名前を指定します。

次の例は、IDXNAME=データセットオプションを使用して、WHERE 式を最適化するために特定のインデックスを使用することを指定します。SAS System は、データファイルを順番に検索する方がリソース効率がよくなる場合でも、それを無視します。SAS System は、指定されたインデックスが最良のものであるかどうかの判定は試みません。(次の例では、EMPNUM インデックスは、NOMISS オプションを使用して作成されていないことに注意してください)。

```
data mydata.empnew;
   set mydata.employee (idxname=empnum);
   where empnum < 2000;
```

詳細については、*SAS データセットオプション: リファレンス*にある IDXNAME データセットオプションの説明を参照してください。

注: IDXWHERE=と IDXNAME=は相互に排他的であり、両者を一緒に使用することはできません。両オプションを同時に指定するとエラーになります。

SAS ログにインデックスの使用情報を表示する

インデックスの使用に関する情報を SAS ログ内に表示するには、MSGLEVEL=システムオプションの値を、デフォルト値の N から I に変更します。**options msglevel=i;**を実行すると、次のように動作します。

- インデックスが使用される場合は、インデックスの名前を示すメッセージを表示します。
- インデックスが使用されない場合で、WHERE 式の条件を少なくとも 1 つ最適化できるインデックスが存在するときは、SAS System にインデックスを使用させるためにはどうすればよいかを提案するメッセージを表示します。たとえば、データファイルをインデックス順に並べ替えることや、より多くのバッファを指定することを提案するようなメッセージを表示します。
- 設定によってインデックス処理に影響が出る可能性がある場合は、メッセージに IDXWHERE=データセットオプションまたは IDXNAME=データセットオプションの設定値が表示されます。

SAS ビューでのインデックスの使用

SAS ビューは SAS データセットの一種であり、他のファイルからデータ値を取得します。SAS ビューには、DATA ステップビューと PROC ビューの 2 種類があります。SAS ビューの詳細については、27 章, “SAS ビュー” (665 ページ)を参照してください。

SAS ビューではインデックスを作成できません。インデックスの作成対象となる SAS データセットは、データファイルでなければなりません。ただし、SAS ビューがインデックス付きデータファイルから作成されている場合は、インデックスを使用することができます。つまり、ビュー定義がキー変数を使用した WHERE 式を含む場合、SAS System はインデックスの使用を試みます。また、SAS ビューの使用時にキー変数を利用する別の方法もあります。

この例では、キー変数 State を持つデータファイル Crime から、Stat という名前の SQL ビューを作成します。さらに、ビュー定義には WHERE 式も指定されています。

```
proc sql;
   create view stat as
   select * from crime
   where murder > 7;
quit;
```

SQL ビューを参照する次の PRINT プロシジャを、キー変数 State を指定した WHERE ステートメントと一緒に実行しても、インデックスを使用して WHERE ステートメントを最適化することはできません。SQL ビューでは、ビューで定義された WHERE 式と、他のプロシジャ、DATA ステップ、SCL で指定された WHERE 式の条件式を結合して使用することはできません。

```
proc print data=stat;
   where state > 42;
run;
```

ただし、キー変数 State を指定した WHERE 句を含む SQL プロシジャを実行する場合、SQL ビューでは 2 つの WHERE 式の条件を結合して使用することができ、それによって SAS System はインデックス State を使用できるようになります。

```
proc sql;
select * from stat where state > 42;
quit;
```

BY 処理でのインデックスの使用

BY 処理を使用すると、BY ステートメントで指定された変数の値に基づく特定の順序で、オブザベーションを処理できます。データファイルにインデックスを付けると、データファイルを並べ替えることなく、BY ステートメントを使用できます。1 つ以上の変数に基づいてインデックスを作成すると、オブザベーションを数値または文字の昇順で処理できます。使用方法は、単純に、BY ステートメントで、インデックス付き変数またはそのリストを指定します。

たとえば、変数 LastName にインデックスがある場合、次の BY ステートメントは、インデックスを使用して、変数 LastName の昇順に値を処理します。

```
proc print;
   by lastname;
```

BY ステートメントを指定すると、SAS System は適切なインデックスを検索します。インデックスが存在する場合、データファイルからオブザベーションを取得する際に、自動的にインデックスの順序に従います。

次の場合、BY ステートメントはインデックスを使用します。

- BY ステートメントが 1 つの変数で構成され、その変数が単一インデックスのキー変数または複合インデックスの最初のキー変数である場合
- BY ステートメントが複数の変数で構成され、最初の変数が単一インデックスのキー変数または複合インデックスの最初のキー変数である場合

たとえば、変数 Major に単一インデックスがある場合、次の BY ステートメントは、インデックスを使用して、変数 Major の昇順に値を処理します。

```
by major;
by major state;
```

変数 ZipCode および SchoolId で構成される複合インデックス ZipId が存在する場合、次の BY ステートメントはインデックスを使用します。

```
by zipcode;
by zipcode schoolid;
by zipcode schoolid name;
```

ただし、次の BY ステートメントの場合、複合インデックス ZIPID は使用されません。

```
by schoolid;
by schoolid zipcode;
```

また、次の場合、BY ステートメントはインデックスを使用しません。

- BY ステートメントに、DESCENDING オプションまたは NOTSORTED オプションが含まれている場合
- インデックスが NOMISS オプションを使用して作成されている場合
- BY ステートメントで指定する変数に基づいて並べ替えられた順序で、物理的にデータファイルが格納されている場合

注: インデックスを使用して BY ステートメントを処理しても、データファイルを単純に並べ替えた場合よりも効率的でない場合があります。これは、ページあたりのオブザベーションのブロック数が高いデータファイルに特に当てはまります。一般的に、BY ステートメントに対してインデックスを使用するのは利便性のためであり、必ずしもパフォーマンスが向上するためではありません。

WHERE とBY の両方の処理でのインデックスの使用

WHERE 式と BY ステートメントの両方を指定すると、両方の必要条件を満たす 1 つのインデックスが検索されます。そのようなインデックスが見つからない場合は、BY ステートメントが優先されます。

ただし、最適化によって BY ステートメントの順序が無効になる場合は、インデックスを使用して WHERE 式を最適化することはできません。たとえば、次のステートメントは、変数 LastName のインデックスを使用して WHERE 式を最適化することができます。これは、インデックスが返すオブザベーションの順序と、BY ステートメントが要求する順序が競合しないためです。

```
proc print;
  by lastname;
  where lastname >= 'Smith';
run;
```

一方、次のステートメントでは、WHERE 式の最適化に変数 LastName のインデックスを使用できません。これは、変数 EmpId の昇順で戻されるオブザベーションを BY ステートメントが要求するためです。

```
proc print;
   by empid;
   where lastname = 'Smith';
run;
```

SET ステートメントと MODIFY ステートメントに KEY=オプションを用いてインデックスを指定する

SET ステートメントと MODIFY ステートメントでは、KEY=オプションを使用することによって、DATA ステップでインデックスを指定して、データファイルの特定のオブザベーションを取得することができます。

次の MODIFY ステートメントは、データファイル Invty.Stock には変数 Partno のインデックスがあることを前提とした、KEY=オプションの使用方法を示しています。KEY=オプションを使用すると、SAS System は、インデックスを使用して、特定のオブザベーションを直接取得します。

```
modify invty.stock key=partno;
```

注: KEY=オプションを含む DATA ステップ内で BY ステートメントを使用することはできません。また、KEY=オプションを含むデータファイルに対して WHERE 処理を行うことはできません。

インデックスの活用

通常ではインデックスを使用しないアプリケーションを、インデックスを利用するように書き直すことができます。次にその例を示します。

- インデックスを使用することがないサブセット化 IF ステートメントを、WHERE ステートメントで置き換えることができます。

  注意:
  ただし、IF および WHERE ステートメントは処理方法が異なるため、SET、MERGE、UPDATE ステートメントを使用する DATA ステップでは結果が異なる場合があるので、注意が必要です。 これは、WHERE ステートメントがオブザベーションを選択する時期が、オブザベーションがプログラムデータベクトル(PDV)に読み込まれる

前であるのに対して、サブセット化 IF ステートメントがオブザベーションを選択する時期は、オブザベーションが PDV に読み込まれた後であるためです。

- SEARCH コマンドおよび FIND コマンドのかわりに、FSEDIT プロシジャで WHERE コマンドを使用することができます。

インデックスを保持するプロシジャと SAS 操作

データファイル情報の表示

CONTENTS プロシジャ(または PROC DATASETS の CONTENTS ステートメント)は、次のようなタイプの情報を表示します。

- データファイルのインデックス数およびインデックス名
- キー変数名
- キー変数の有効オプション
- データファイルのページサイズ
- データファイルのページ数
- 百分位数情報(CENTILES オプションを使用)
- インデックスファイルが使用するディスク領域のサイズ

注: 利用可能な情報は、動作環境によって異なります。

アウトプット 26.5 CONTENTS プロシジャの出力

The SAS System

The CONTENTS Procedure

Data Set Name	MYFILES.STAFF	Observations	148
Member Type	DATA	Variables	6
Engine	V9	Indexes	2
Created	08/17/2012 11:10:03	Observation Length	63
Last Modified	08/17/2012 11:16:51	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	8192
Number of Data Set Pages	3
First Data Page	1
Max Obs per Page	129
Obs in First Data Page	104
Index File Page Size	4096
Number of Index File Pages	5
Number of Data Set Repairs	0
Filename	C:\My Documents\staff.sas7bdat
Release Created	9.0401B0
Host Created	W32_7PRO

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
4	city	Char	15
3	fname	Char	15
6	hphone	Char	12
1	idnum	Char	4
2	lname	Char	15
5	state	Char	2

Alphabetic List of Indexes and Attributes

#	Index	Unique Option	NoMiss Option	# of Unique Values	Variables
1	idnum	YES	YES	148	
2	name			148	fname lname

インデックス付きデータファイルのコピー

COPY プロシジャ(または DATASETS プロシジャの COPY ステートメント)を使用して、インデックス付きデータファイルをコピーする場合は、新しいデータファイルのインデックスファイルを再作成するかどうかを、INDEX=YES|NO オプションで指定できます。デフォルトは YES で、インデックスが再作成されます。ただし、インデックスの再作成は、COPY プロシジャ(PROC COPY ステップ)の処理時間を増大させます。

ディスクからディスクにコピーする場合、インデックスは再作成されます。ディスクからテープにコピーする場合、インデックスはテープ上には再作成されません。ただし、ディスクからテープにコピーした後で、テープからディスクにコピーすると、インデックスを再作成できます。COPY プロシジャの MOVE オプションを使用してデータファイルを移動すると、インデックスファイルは IN=オプションに指定したライブラリから削除され、OUT=オプションに指定したライブラリに再作成されます。

CPORT プロシジャにも INDEX=オプションがあります。このオプションを使用すると、インデックス付きデータファイルをエクスポートする場合に、インデックスを移送するかどうかを指定できます。デフォルトでは、CPORT プロシジャは、インデックス付きデータファイルを移送する場合に、インデックスを移送します。ただし、CIMPORT プロシジャはインデックスファイルを処理しないため、インデックスを再作成する必要があります。

インデックス付きデータファイルの更新

インデックス付きデータファイルの値が追加、変更、削除されるたびに、インデックスは自動的に更新されます。インデックスに対して影響を与えるデータファイルの操作を、次の表に示します。

表 26.10 *保守タスクとインデックスの結果*

タスク	結果
データセットを削除する場合	インデックスファイルが削除されます。
データセットの名前を変更する場合	インデックスファイルの名前が変更されます。
キー変数の名前を変更する場合	単一インデックスの名前が変更されます。
キー変数を削除する場合	単一インデックスが削除されます。
オブザベーションを追加する場合	インデックスエントリが追加されます。
オブザベーションを削除する場合	インデックスエントリが削除され、空き領域が再利用のために回復されます。
オブザベーションを更新する場合	インデックスエントリが削除され、新しいインデックスエントリが挿入されます。

注: データセットの追加、修正、削除を実行する場合は、SAS System を使用して行ってください。動作環境のコマンドを使用してこれらの操作を行うと、ファイルが使用できなくなります。

インデックス付きデータファイルの並べ替え

インデックス付きデータファイルの並べ替えができるのは、SORT プロシジャの出力先を新しいデータファイルに設定して、元のデータファイルをそのまま残す場合だけです。ただし、新しいデータファイルに対して自動的にインデックスは付けられません。

注: FORCE オプションを使用してインデックス付きデータファイルを並べ替えると、インデックスファイルが削除されます。

インデックス付きデータファイルへのオブザベーションの追加

インデックス付きデータファイルにオブザベーションを追加するには、別の処理が必要になります。SAS System は、インデックスの値とデータファイルの値との一貫性を自動的に保持します。

値の重複

重複する同一値があるデータファイルでは、UNIQUE オプションを使用しないでインデックスを作成します。そのようなインデックスの場合、1 つの値に対して複数の RID が生成されます。多くの重複する同一値がある大きなデータファイルの場合、特定の値の RID のリストがインデックスファイルの数ページにも及ぶ場合があります。RID は物理順に格納されるため、特定の値を含むデータファイルに追加された新しいオブザベーションは、RID のリストの末尾に格納されます。つまり、RID のリストの末尾を検出するためにインデックス内を探索することにより、入出力動作の回数が増加します。

SAS System は、インデックスの前回の位置を記憶しています。これにより、同一値をさらに挿入する場合に、RID のリストの末尾をすぐに検出します。

インデックス付きデータファイルへのデータの追加

SAS System では、インデックス付きデータファイルにデータを追加する際のパフォーマンスの向上を実現しています。SAS System は、すべてのオブザベーションが追加されるまでインデックスの更新を保留し、その後、新しく追加されたオブザベーションのデータを含むインデックスを更新します。詳細については、*Base SAS Procedures Guide* にある DATASETS プロシジャの APPEND ステートメントの説明を参照してください。

破損したインデックスの修復

インデックスは、データファイルやカタログが破損するのと同様に破損する可能性があります。データファイルが破損した場合、DATASETS プロシジャで REPAIR ステートメントを使用して、データファイルの修復や、欠損したインデックスの再作成を行います。次にその例を示します。

```
proc datasets library=mylib;
   repair mydata;
run;
```

インデックスと一貫性制約

一貫性制約でもインデックスを使用できます。インデックスを使用する一貫性制約を作成する場合、適切なインデックスがすでに存在するならば、そのインデックスが使用されます。適切なインデックスが存在しない場合、新しいインデックスが作成されます。インデックスを作成すると、そのインデックスは作成者(ユーザーまたは一貫性制約のいずれか)により所有されているものとしてマークされます。

ユーザーまたは一貫性制約のいずれかが、すでに存在するインデックスの作成を要求した場合は、その要求者にも当該インデックスの所有者としてのマークが付けられます。インデックスが両者により所有されている場合、所有者のどちらかがそのインデックスの削除を要求すると、所有者として要求を行った者のみが削除されます。インデックスは、一貫性制約とユーザーの両者がそのインデックスの削除を要求した場合にのみ削除されます。インデックスを削除できなかった場合は、SAS ログに情報が表示されます。

インデックスと CEDA 処理

CEDA を使用して SAS データファイルを処理する場合、インデックスはサポートされません。たとえば、定義されたインデックスを持つ SAS データファイルを Windows 環境から UNIX 環境へと移動する場合、CEDA はユーザーのためにファイルを自動的に変換しますが、インデックスは使用できません。したがって、インデックスファイルを使用した WHERE 句の最適化はサポートされません。

CEDA 処理の詳細については、32 章, “クロス環境データアクセス(CEDA)を用いたデータ処理” (707 ページ)を参照してください。

拡張属性

定義

拡張属性は、SAS ファイルのカスタマイズされたメタデータと見なすことができます。データセットのラベル、変数の長さやラベルなどの SAS 属性は、SAS システム属性としてあらかじめ定義されていますが、拡張属性はユーザーが定義する属性です。それらは一般的に、名前と値のペアで編成され、SAS データセット内の変数または SAS データセットに関連付けられます。拡張属性は、(名前、値)ペアで BASE Engine に対応して編成されます。これらのデータは、ファイル拡張子 sas7bxat が付いた別々の SAS データファイルに格納されます。

拡張属性の有効化と操作

さまざまな XATTR ステートメントを DATASETS プロシジャで使用して、拡張属性のオプションを作成、追加、削除、更新、除外、指定することができます。PROC CONTENTS を使用しても、データセットや変数拡張属性を表示できます。拡張属性の詳細については、“Extended Attributes” (*Base SAS Procedures Guide*)を参照してください。

次の出力では、拡張属性を持つ SAS データセットに対する PROC CONTENTS の実行結果を示します。

アウトプット 26.6 *拡張属性のリストを表示する PROC CONTENTS 出力*

Alphabetic List of Data Set Extended Attributes

Extended Attribute	Numeric Value	Character Value
attrib	.	table
role	.	train

Alphabetic List of Extended Attributes on Variables

Extended Attribute	Attribute Variable	Numeric Value	Character Value
level	income	.	interval
level	purchase	.	nominal
role	age	.	reject
role	income	.	input
role	purchase	.	target

データファイルの圧縮

圧縮の定義

ファイルを圧縮すると、各オブザベーションを表すために必要となるバイト数を減らすことができます。圧縮されたファイルでは、各オブザベーションは可変長レコードとなります。一方、圧縮されていないファイルでは、各オブザベーションは固定長レコードとなります。

ファイル圧縮のメリットとしては次のことが挙げられます。

- ファイルのストレージ要件を下げることができる
- 処理中にデータの読み書きに必要となる入出力操作を減らすことができる

ファイル圧縮にはデメリットもあります。次にその例を示します。

- 各オブザベーションを非圧縮化するオーバーヘッドがかかるため、圧縮ファイルの読み込みにはより多くの CPU リソースが必要となる
- 状況によっては、圧縮の結果生成されたファイルのサイズが逆に増える場合がある

圧縮の要求

デフォルトでは、SAS データファイルは圧縮されません。圧縮を行うには、次のオプションを使用します。

- COMPRESS=システムオプションを使用すると、SAS セッション中に作成されるすべてのファイルを圧縮できます
- LIBNAME ステートメントの COMPRESS=オプションを使用すると、特定の SAS ライブラリのすべてのデータファイルを圧縮できます
- COMPRESS=データセットオプションを使用すると、個々のデータセットを圧縮できます

データファイルを圧縮するには、次のいずれかを指定します。

- RLE (Run Length Encoding)圧縮アルゴリズムを使用する場合、COMPRESS=CHAR を指定します
- RDC (Ross Data Compression)アルゴリズムを使用する場合、COMPRESS=BINARY を指定します

圧縮データファイルを作成すると、同ファイルの圧縮で得られる縮小の割合を示す情報が SAS ログに表示されます。圧縮の割合は、圧縮データセットのサイズと非圧縮データセットのサイズとを比較することで算出されます。

ファイルが圧縮された後は、その設定がファイルの永久属性です。これは、設定を変更するにはファイルを再作成する必要があることを意味しています。ファイルを解凍するには、圧縮データファイルをコピーする DATA ステップで COMPRESS=NO オプションを指定します。

COMPRESS=データセットオプションの詳細については、*SAS データセットオプション: リファレンス*を参照してください。LIBNAME ステートメントの COMPRESS=オプションの詳細については、*SAS Statements: Reference* を参照してください。COMPRESS=システムオプションの詳細については、*SAS System Options: Reference* を参照してください。

圧縮要求の無効化

ファイルを圧縮すると、固定長のデータブロックが各オブザベーションに追加されます。この追加データブロック(オブザベーション当たり、32 ビットホストの場合 12 バイト、64 ビットホストの場合 24 ビットが追加される)のために、ファイルによっては、圧縮後にファイルサイズが増加するものもあります。たとえば、非常に短いレコード長を持つファイルの場合、圧縮するとファイルサイズが増加することがあります。

データセットの圧縮が要求されると、SAS System は、圧縮によってファイルのサイズが増加するかどうかを判定するよう試みます。SAS System は、変数の長さを調べます。変数の数や変数長が原因で、圧縮ファイルのサイズを、圧縮しない場合よりも、オブザベーション当たり最低 12 バイト(32 ビットの場合)または 24 バイト(64 ビットの場合)より小さくできない場合、圧縮は無効になり、メッセージが SAS ログに書き出されます。

SAS System によって、圧縮することにより圧縮しない場合よりもサイズを小さくできないと判定される単純なデータセットの例を次に示します。

```
data one (compress=char);
   length x y $2;
   input x y;
   datalines;
ab cd
```

```
;
```

次の出力が SAS ログに書き出されます。

ログ 26.1 *圧縮要求を無効化した場合の SAS ログ出力*

```
NOTE:Compression was disabled for data set WORK.ONE because compression overhead
would increase the size of the data set.NOTE:The data set WORK.ONE has 1
observations and 2 variables.
```

32 ビット SAS データファイルのオブザベーションカウントの拡張

拡張オブザベーションカウントの定義

拡張オブザベーションカウントとは、32 ビット長整数で表される最大数を超えてオブザベーションをカウントする SAS データファイルの属性です。SAS 9.4 では、出力 32 ビット SAS データファイルを作成する場合に、結果として生じる拡張ファイル形式は、カウンタに関しては 64 ビットファイルのように動作します。しかしながら、拡張オブザベーションカウントを使用できる SAS データファイルは、SAS 9.3 より前のリリースとは互換性がありません。

最大オブザベーションカウントの詳細については、“SAS データファイルのオブザベーションカウントについて” (600 ページ)を参照してください。

拡張オブザベーションカウントについて

従来、BASE Engine には、32 ビット長整数を使用する動作環境でカウントとフルサポートが可能なオブザベーションは最大約 20 億までという制限がありました。SAS 9.3 では機能が追加され、この制限を拡張して 64 ビット長整数を使用する動作環境の制限と一致させられるようになりました。出力 SAS データファイルのオブザベーションカウントを拡張するために、EXTENDOBSCOUNTER=データセットオプションおよび LIBNAME ステートメントオプションが提供されており、デフォルトでは NO に設定されていました。拡張オブザベーションカウントの付いた SAS データファイルを SAS 9.3 で作成するには、SAS データセットオプションまたは LIBNAME ステートメントオプションのいずれかで、EXTENDOBSCOUNTER=YES を指定します。

SAS 9.4 では、拡張オブザベーションカウント機能の拡張により、拡張オブザベーションカウントの付いた 32 ビット SAS データファイルが自動的に作成され、EXTENDOBSCOUNTER=システムオプションが提供されます。SAS 9.4 では、EXTENDOBSCOUNTER=データセットオプション、LIBNAME ステートメントオプション、およびシステムオプションは、デフォルトで YES に設定されています。したがって、SAS データファイルが拡張オブザベーションカウントで作成されないようにするには、SAS 9.4 では、EXTENDOBSCOUNTER=NO を指定する必要があります。

EXTENDOBSCOUNTER=オプションの使用

出力 SAS データファイルでオブザベーションカウントの拡張を制御するには、EXTENDOBSCOUNTER=オプションを使用します。このオプションを指定すると、32 ビット長で表される最大数を超えてオブザベーションをカウントするような拡張ファイル形式を要求できます。

注: EXTENDOBSCOUNTER=オプションには、別名 EOC=があります。

EXTENDOBSCOUNTER=オプションは次の場合にサポートされます。

- SAS データセットオプションとして、EXTENDOBSCOUNTER=は、そのオプションを付けて新規作成された出力ファイルに適用されます。データセットオプションを使用して、LIBNAME ステートメントとシステムオプションの設定を上書きできます。詳細については、“EXTENDOBSCOUNTER= Data Set Option” (*SAS Data Set Options: Reference*)を参照してください。
- LIBNAME ステートメントオプションとして、EXTENDOBSCOUNTER=は、SAS ライブラリに新規作成された出力ファイルに適用されます。詳細については、“LIBNAME Statement” (*SAS Statements: Reference*)を参照してください。
- システムオプションとして、EXTENDOBSCOUNTER=は、SAS セッションで新規作成される出力ファイルに影響を及ぼします。詳細については、“EXTENDOBSCOUNTER= System Option” (*SAS System Options: Reference*)を参照してください。

出力 SAS データファイルでオブザベーションカウントが拡張されないように要求するには、EXTENDOBSCOUNTER=NO を指定する必要があります。たとえば、32 ビット長整数を使用する動作環境で次のプログラムを実行すると、32 ビット長で表される最大数を超えるオブザベーションカウントに拡張されていない出力 SAS データファイルが作成されます。この例では、EXTENDOBSCOUNTER=オプションを DATA ステートメント内のデータセットオプションとして指定しています。SAS 9.3 より前の SAS バージョンでファイルを処理する必要がある場合、出力 SAS データファイルでオブザベーションカウントが拡張されないように要求することが必要な場合があります。

```
libname myfiles 'C:\MyFiles';

data myfiles.bigfile (extendobscounter=no);
   .
   .
   .
run;
```

EXTENDOBSCOUNTER=を LIBNAME ステートメントオプションとして指定すると、SAS ライブラリに作成されるすべてのファイルの拡張ファイル形式を制御できます。次のプログラムでは、COPY プロシジャを使用して、SAS ライブラリ内のすべてのファイルを拡張オブザベーションカウントなしで再作成します。

```
libname new 'C:\NewFiles' extendobscounter=no;
libname old 'C:\OldFiles';

proc copy in=old out=new;
run;
```

最大オブザベーションカウントを超過したデータファイルの回復

カウントできるオブザベーションの最大数を超過した SAS データファイルを回復するには、その SAS データファイルを単純に SAS 9.4 で再作成します。最大オブザベーションカウントを超過したファイルを変換する方法としては、DATA ステップの SET ステートメントを使うことが挙げられます。たとえば、次のプログラムでは、既存のファイルをコピーすることにより、拡張オブザベーションカウントが付いた新しいファイルを作成します。デフォルトでは、EXTENDOBSCOUNTER=YES です。

```
libname lib 'C:\Myfiles';

data lib.b;
   set lib.a;
run;
```

拡張オブザベーションカウントを示すファイル属性

拡張オブザベーションカウントを付けて SAS データファイルを作成すると、そのファイルには、オブザベーションカウントを拡張するためのファイル形式の拡張を示す属性が含められます。たとえば、次の CONTENTS プロシジャ出力では、**ExtendObsCounter** 情報が **Engine/Host Dependent Information** の下に表示されます。

注: SAS データファイルに拡張オブザベーションカウントを示すファイル属性が含まれていない場合、**ExtendObsCounter** フィールドは表示されません。

アウトプット 26.7 *ExtendObsCounter 属性を表示する CONTENTS プロシジャ出力*

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	2039
Obs in First Data Page	2
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\My Documents\bigfile.sas7bdat
Release Created	9.0401M0
Host Created	X64_7PRO

拡張オブザベーションカウントの動作に関する注意点

拡張オブザベーションカウント属性を持つ SAS データファイルは SAS 9.3 でサポートが開始されました。拡張ファイル形式は、SAS 9.3 より前のリリースでは互換性がありません。拡張オブザベーションカウント属性を持つ SAS データファイルを SAS 9.3 より前のリリースで開こうとするとエラーメッセージが出されます。次にその例を示します。

```
ERROR:File MYFILES.EXTEND.Data not compatible with this SAS version.
```

拡張オブザベーションカウント属性は、新しいファイルには継承されません。ただし、EXTENDOBSCOUNTER=はデフォルトで YES に設定されているため、拡張オブザベーションカウント属性を持つファイルから新しいファイルを作成する際に、オブザベーションカウントの拡張を明示的に要求する必要はありません。たとえば、次のプログラムでは、既存の Extend1 というファイルから Extend2 という名前のファイルを新しく作成します。Extend1 は拡張オブザベーションカウント属性を持っています。拡張オブザベー

ションカウント属性は新しいファイルに継承はされませんが、新しいファイルの DATA ステートメントには EXTENDOBSCOUNTER=YES データセットオプションを指定する必要はありません。EXTENDOBSCOUNTER=はデフォルトで YES に設定されています。

```
data extend2;
   set extend1;
run;
```

逆に、拡張オブザベーションカウントが付いていないファイルを、拡張オブザベーションカウント属性のないファイルから作成する場合、新しいファイルには EXTENDOBSCOUNTER=NO オプションを指定する必要があります。たとえば、次のプログラムでは、NoExtend2 という新しいファイルが既存の NoExtend1 という名前のファイルから作成されます。既存のファイルには、拡張オブザベーションカウント属性はありません。新しいファイルの DATA ステートメントには、EXTENDOBSCOUNTER=NO データセットオプションを指定する必要があります。

```
data noextend2 (eoc=no);
   set noextend1;
run;
```

拡張オブザベーションカウント属性により影響を受ける操作としては、次のものが挙げられます。

- ファイルをコピーする SAS 機能(APPEND プロシジャ、COPY プロシジャ、MIGRATE プロシジャ、SET ステートメントなど)は、拡張オブザベーションカウント属性をコピーしません。
- SAS/SHARE クライアントセッションにおいて、LIBNAME ステートメントで EXTENDOBSCOUNTER=オプションを SERVER=オプションと一緒に指定した場合、同オプションは無視されます。
- 拡張オブザベーションカウントを付けて作成した SAS データファイルに、FIRSTOBS=オプションまたは OBS=オプションを指定する場合、32 ビット環境で 2^{31}-1 以上のオブザベーションを含むファイルでパフォーマンスが向上します。

64 ビット動作環境でのオブザベーションカウントの拡張

オブザベーションカウントを 64 ビット長整数として格納する動作環境では、オブザベーションカウントの拡張は無視されます。たとえば、64 ビット長整数を扱う動作環境(例: 64 ビット Itanium 上の HP-UX)で次のプログラムを実行すると、SAS ログに次のようなメッセージが表示されます。

```
libname myfiles '/u/myid/myfiles';

data myfiles.bigfile (extendobscounter=yes);
   .
   .
   .
run;

   NOTE:EXTENDOBSCOUNTER=YES is ignored on a SAS data set with a 64-bit long observation co
```

ただし、OUTREP=オプションを使用して 32 ビットデータ表現を含む出力ファイルを作成する場合、オブザベーションカウントは 32 ビット長で格納されます。この場合、カウンタに関して 64 ビットファイルと同様に動作するような 32 ビットファイルを作成するために、オブザベーションカウントの拡張がサポートされます。たとえば、次のプログラム

を 64 ビット Itanium 上の HP-UX 環境でサブミットすると、Microsoft Windows 64 ビットエディションのデータ表現を含む出力ファイルが作成されますが、同ファイルでは 32 ビット長整数型が使用されます。

```
libname myfiles '/u/MyFiles';

data myfiles.bigfile (outrep=windows_64);
   .
   .
   .
run;
```

ヒント Microsoft Windows 64 ビットエディションは 64 ビット対応の動作環境ですが、オブザベーションカウントは 32 ビット長整数として格納されます。このため、Microsoft Windows 64 ビットエディション環境では、SAS データファイルのオブザベーションカウントの拡張により、32 ビット長整数で表される最大数を超えてオブザベーションをカウントできるようになります。

オブザベーションカウント拡張がサポートされる場合

オブザベーションカウントの拡張は、オブザベーションカウントを 32 ビット長整数として格納する内部データ表現を持つ出力 SAS データファイルにのみ影響します。

また、オブザベーションカウントを 64 ビット長整数として格納する動作環境では、OUTREP=オプションを指定して 32 ビットデータ表現を持つ出力ファイルを作成している場合を除き、EXTENDOBSCOUNTER=YES は無視されます。

また、オブザベーションカウントを 32 ビット長整数として格納する動作環境であっても、OUTREP=オプションを指定して 64 ビットデータ表現を持つ出力ファイルを作成している場合には、EXTENDOBSCOUNTER=YES は無視されます。

次の表に、SAS 9.4 の動作環境、各動作環境のデータ表現値、デフォルトのオブザベーションカウントのサイズ、オブザベーションカウントの拡張をサポートするかどうかを示します。たとえば、使用する動作環境が 64 ビットプラットフォームの AIX である場合、EXTENDOBSCOUNTER=YES は無効です。ただし、AIX (64 ビットプラットフォーム)で、OUTREP=オプションを使用してデータ表現値を WINDOW_64 に指定する場合は、EXTENDOBSCOUNTER=YES がサポートされます。

表 26.11 *EXTENDOBSCOUNTER=YES を指定する場合*

動作環境	データ表現値	デフォルトのオブザベーションカウントサイズ	EOC=YES
64 ビットプラットフォーム上の AIX	RS_6000_AIX_64	64 ビットカウンタ	無効
Itanium 向け HP-UX (64 ビットプラットフォーム上)	HP_IA64	64 ビットカウンタ	無効
x64 向け Linux	LINUX_X86_64	64 ビットカウンタ	無効
32 ビットプラットフォーム上の Microsoft Windows	WINDOWS_32	32 ビットカウンタ	サポートあり

動作環境	データ表現値	デフォルトのオブザベーションカウントサイズ	EOC=YES
Microsoft Windows の 64 ビットエディション	WINDOWS_64	32 ビットカウンタ	サポートあり
Solaris on SPARC (64 ビットプラットフォーム)	SOLARIS_64	64 ビットカウンタ	無効
z/OS 上の 32 ビット版 SAS	MVS_32	32 ビットカウンタ	サポートあり

27 章

SAS ビュー

SAS ビューの定義

SAS ビューは SAS データセットの一種であり、他のファイルからデータ値を取得します。SAS ビューには、変数(列)のデータ型やデータ長などのディスクリプタ情報のみが含まれています。また、他の SAS データセットや、他のソフトウェアベンダのファイル形式で格納されているファイルからデータ値を取り出すのに必要となる情報も含んでいます。SAS ビューは、メンバタイプ VIEW を持つ SAS ファイルとして管理されます。ほとんどの場合、SAS ビューは、SAS データファイルを扱うのと同様に使用できます。

SAS ビューには、次の 2 種類があります。

ネイティブビュー
: DATA ステップまたは SQL プロシジャを使用して作成する SAS ビュー(DATA ステップビュー、PROC SQL ビュー)です。

インターフェイスビュー
: SAS/ACCESS ソフトウェアを使用して作成する SAS ビューです。DB2 や Oracle などのデータベース管理システム(DBMS)との間でデータを読み書きできます。イン

ターフェイスビューは、SAS/ACCESS ビューとも呼びます。SAS/ACCESS ビューを使用するには、SAS/ACCESS ソフトウェアのライセンスが必要です。

注: SAS/ACCESS ソフトウェアの動的 LIBNAME Engine を利用すると、DBMS のデータにアクセスするネイティブビューを作成できます。詳細については、“SAS/ACCESS ビュー” (673 ページ)か、または使用している DBMS に対応する SAS/ACCESS ドキュメントを参照してください。

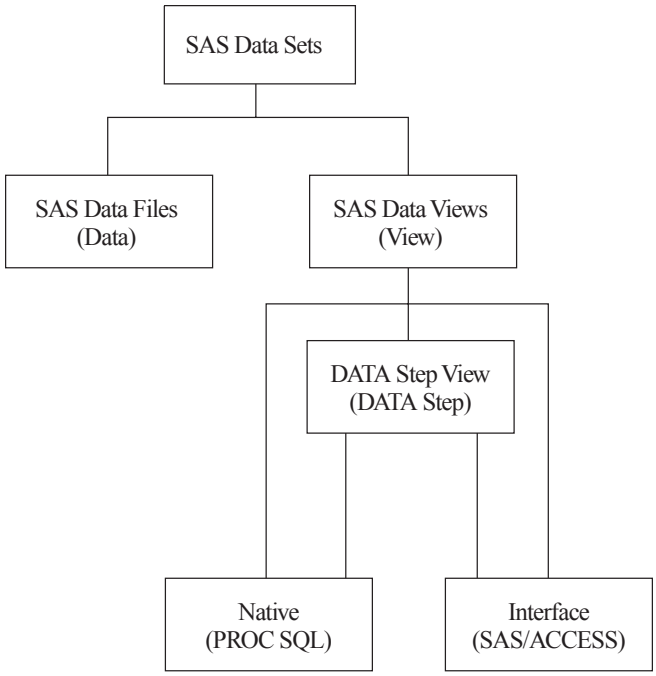
SAS ビューを使用する利点

SAS ビューには、次のような利点があります。

- SAS ビューにより、複数のデータセット(テーブル)を結合できるので、複数の DATA ステップを使用して共通変数に基づく SAS データセットの結合(マージ)を行う必要がありません。
- SAS ビュー定義を格納することで、ディスク領域を節約できます。ビュー定義には、データの検索場所とフォーマットに関する指定だけが格納され、実際のデータは格納されません。
- SAS ビューは実行時にデータを動的に取得するため、常に現時点での入力データセットの情報を得ることができます。
- SAS ビューでは多くの入力ソースからデータを選択できるため、SAS ビューを作成しておけば、他にプログラミングを行わなくても、パッケージ化した一定の情報を利用者に提供できます。
- SAS ビューを使用すると、データ設計の変更がユーザーに与える影響を軽減することができます。たとえば、SAS ビューに格納されているクエリを変更しても、同ビューの結果の特性は変わりません。
- SAS/CONNECT ソフトウェアを使用すれば、異なるホストコンピュータ上にある SAS データセットを結合して、散在する企業データから統合した情報を表示できます。

次の図は、ネイティブビューとインターフェイスビュー、および SAS データファイルとの関係を示しています。

図 27.1 ネイティブ SAS ビューとインターフェイス SAS ビュー



SAS ビューには、次のような使用方法があります。

- 他の DATA ステップまたは PROC ステップへの入力として使用できます。
- SAS データファイルまたは SAS System でサポートされるデータベース管理システム(DBMS)に、データを動的にコピーするために使用します。
- SQL プロシジャを使用する他の入力データソースと組み合わせて使用します。
- SAS/ASSIST ソフトウェア上で、連結したデータセットとして使用します。データの格納方法を意識することなく、データ管理、分析、レポート作成のタスクを実行できます。

SAS ビューを使用する場合の注意点

SAS データファイルと SAS ビューのどちらが目的に適しているかを決定するために考慮するべき点は次のとおりです。

- SAS データファイルはディスク領域を多めに消費し、SAS ビューは処理時間が長めにかかります。
- SAS データファイルの変数については、使用前に並べ替えたりインデックスを作成したりできます。一方、SAS ビューは、既存の形式のままデータを処理します。

DATA ステップビュー

DATA ステップビューの定義

DATA ステップビューは、SAS ビューの中で最も柔軟なデータ処理を行えるネイティブビューです。DATA ステップビューには、次のさまざまな入力データソースからデータを読み込むことのできる、ストアド DATA ステッププログラムが格納されます。

- 生データファイル
- SAS データファイル
- PROC SQL ビュー
- SAS/ACCESS ビュー
- DB2 データや Oracle データなどの DBMS データ

DATA ステップビューの作成

DATA ステップビューを作成するには、DATA ステートメントで指定する最後のデータセット名の後ろに、/ (スラッシュ)を付けて VIEW=オプションで DATA ステップビュー名を指定します。VIEW=オプションは、ソースプログラムをコンパイルし、それを実行せずに、このオプションで指定された DATA ステップビューにコンパイル済みプログラムを格納します。

たとえば、次のステートメントは Dept.A という名前の DATA ステップビューを作成します。

```
libname dept 'SAS-library';

data dept.a / view=dept.a;
   … more SAS statements …
run;
```

SAS ビューが SAS ライブラリ内に存在する場合に、同じメンバ名を使用して新しいビュー定義を作成すると、古い SAS ビューが上書きされます。

SAS バージョン 8 から、DATA ステップビューでソースステートメントが保存されるようになりました。ソースステートメントを SAS ログに表示するには、DESCRIBE ステートメントを使用します。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て、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;
```

SAS ビューの作成方法と DESCRIBE ステートメントの使用方法の詳細については、*SAS Statements: Reference* を参照してください。

DATA ステップビューの機能

DATA ステップを使用すると、次の操作を行えます。

- INPUT ステートメントで読み込むことのできるファイルを直接処理します。
- 他の SAS データセットを読み込みます。

- 外部ファイルなどの入力データソースを使用したり中間 SAS データファイルを作成したりせずに、データを生成します。

DATA ステップビューは DATA ステップで生成されるため、外部ファイルや既存の SAS データセットなど、さまざまな入力データソースから取得した入力データを操作および管理できます。DATA ステップビューは、他の SAS ビューよりも多くの機能を備えています。

DATA ステップビューとコンパイル済みストアド DATA ステッププログラムの違い

DATA ステップビューとコンパイル済みストアド DATA ステッププログラムでは、次の点が異なります。

- DATA ステップビューは、他の DATA ステップまたは PROC ステップによって入力データセットとして参照されたときに、暗黙的に実行されます。主に、呼び出すプロシジャや DATA ステップに、データを一度に 1 オブザベーション(レコード)ずつ提供するために使用されます。
- コンパイル済みストアド DATA ステッププログラムは、DATA ステートメントの PGM=オプションで指定されたとき、明示的に実行されます。コンパイル済みストアド DATA ステッププログラムは通常、SAS データファイルの作成やレポートの作成など、より定型的なタスクを実行するために使用されます。

コンパイル済みストアド DATA ステッププログラムの詳細については、28 章, “コンパイル済みストアド DATA ステッププログラム” (675 ページ)を参照してください。

制限事項と必要条件

グローバルステートメントは、DATA ステップビューには適用されません。.FILENAME、FOOTNOTE、LIBNAME、OPTIONS、TITLE ステートメントなどのグローバルステートメントは、SAS ビューを作成する DATA ステップの中に記述しても、同ビューには作用しません。ソースプログラムのステートメント内にグローバルステートメントを記述した場合、DATA ステップビューは格納されますが、そのグローバルステートメントは格納されません。同ビューを参照しても、意図したとおりにグローバルステートメントが実行されない可能性があります。

ビューを作成すると、それが返す変数のラベルも作成されます。ある DATA ステップビューが変数ラベルを含んでいるデータセットを読み込んだ場合、そのビューが作成された後にラベルを変更したとしても、プロシジャ出力では元のラベルが表示されます。この変更後の新しいラベルをプロシジャ出力で表示するには、当該ビューを再コンパイルする必要があります。

パフォーマンスに関する注意点

- DATA ステップコードは、ユーザーが DATA ステップビューを使用するたびごとに実行されるため、結果として、システムのオーバーヘッドが著しく増える場合があります。また、ステップ間でデータが変化する危険性があります。ただし、これは、最も直近のデータが利用できること、すなわち、ビューのコンパイル時のデータではなく、ビューの実行時のデータが利用できることも意味します。
- データの読み取りや引き渡しの回数が多くなると、処理のオーバーヘッドもそれに応じて増加します。
  - データの引き渡しが 1 回だけ要求された場合には、データセットは作成されません。引き渡しが 1 回要求された場合は、従来の処理方法と比較すると入出力操作の回数と処理時間が少なくなるため、パフォーマンスが向上します。

- ランダムアクセスまたは複数回のデータの引き渡しが要求された場合は、SAS ビューを実行すると、生成されたオブザベーションをすべて含んだ予備ファイルが作成されます。この結果、後続のデータ引き渡しでは、それ以前の引き渡しで読み込まれたデータと同じデータを読み込むことができます。場合によっては、ビューの SPILL=データセットオプションを使用することで、予備ファイルのサイズを縮小できます。

- VBUFSIZE=システムオプションと OBSBUF=データセットオプションを使用すると、DATA ステップビューの処理時に実行速度を向上させることができます。SAS ビューでのパフォーマンス最適化に関する情報については、“SAS DATA ステップビューに対する VBUFSIZE=と OBSBUF=の設定” (191 ページ)を参照してください。

  VBUFSIZE=システムオプションの詳細については、“VBUFSIZE= System Option” (*SAS System Options: Reference*)を参照してください。OBSBUF データセットオプションの詳細については、“OBSBUF= Data Set Option” (*SAS Data Set Options: Reference*)を参照してください。

例 1: データのマージによるレポート作成

複数のファイルからデータを読み込んで結合(マージ)する場合、結合済みデータを格納するファイルを作成する必要がない場合は、結合結果を得るための DATA ステップビューを作成します。この DATA ステップビューは、後続のアプリケーションで使用することができます。

たとえば、次のステートメントは DATA ステップビュー MyV9Lib.QTR1 を定義し、データファイル V9lr.Clothes の売上額とデータファイル V9lr.Equip の売上額をマージします。データファイルは日付に基づいて 1 日分ごとにマージされ、変数 `Total` の値はそれぞれの日について合計額が計算されます。

```
libname myv9lib 'SAS-library';
libname v9lr 'SAS-library';

data myv9lib.qtr1 / view=myv9lib.qtr1;
   merge v9lr.clothes v9lr.equip;
      by date;
   total = cl_v9lr + eq_v9lr;
run;
```

次の PRINT プロシジャにより、作成したビューを出力します。

```
proc print data=myv9lib.qtr1;
run;
```

例 2: 追加の出力ファイルの作成

この例にある DATA ステップでは、学生のデータが格納されている Student という名前の外部ファイルを読み込み、問題のあることが判明したオブザベーションをデータファイル MyV9Lib.Problems に書き出します。また、この DATA ステップでは、DATA ステップビュー MyV9Lib.Class も定義しています。この DATA ステップを実行しても、SAS データファイル MyV9Lib.Class は作成されません。

FILENAME ステートメントと LIBNAME ステートメントはどちらもグローバルステートメントであるため、SAS ビューを定義するプログラムの外側に記述する必要があります。SAS ビューにはグローバルステートメントを格納できません。

外部ファイル Student の内容は次のとおりです。

```
dutterono  MAT   3
```

```
lyndenall  MAT
frisbee    MAT  94
           SCI  95
zymeco     ART  96
dimette         94
mesipho    SCI  55
merlbeest  ART  97
scafernia       91
gilhoolie  ART 303
misqualle  ART  44
xylotone   SCI  96
```

次に、出力データファイルを作成する DATA ステップを示します。

```
libname myv9lib 'SAS-library';
filename student 'external-file-specification';  1

data myv9lib.class(keep=name major credits)
   myv9lib.problems(keep=code date) / view=myv9lib.class; 2
   infile student;
   input name $ 1-10 major $ 12-14 credits 16-18;  3

select;
when (name=' ' or major=' ' or credits=.)
         do code=01;
            date=datetime();
            output myv9lib.problems;
         end; 4
when (0<credits<90)
         do code=02;
            date=datetime();
            output myv9lib.problems;
         end; 5
otherwise
       output myv9lib.class;
   end;
run; 6
```

次の例では、上記のプログラムで作成されたデータファイルを出力する方法を示します。MyV9Lib.Class には、エラーなしで処理された Student からのオブザベーションが含まれています。データファイル MyV9Lib.Problems には、エラーがあるオブザベーションが含まれています。

ソースデータファイル Student でデータが頻繁に変更される場合は、SAS ビューおよび SAS データファイルに戻される結果の値に違いが生じることがあります。

- 前述の例の DATA ステップを実行した時点から、次の例にある PRINT プロシジャを実行する時点までに、入力データファイル Student にオブザベーションが新しく追加され、それにエラーがない場合は、SAS ビュー MyV9Lib.Class に表示されます。
- それに対して、エラーのあるオブザベーションが Student に新しく追加された場合は、DATA ステップをもう一度実行した時点でそのオブザベーションがデータファイル MyV9Lib.Problems に表示されます。

SAS ビューは、入力データファイルが使用されるたびに、動的に更新されます。SAS データファイルは、新しいデータが直接書き込まれた場合を除いて、入力データファイルが使用されるたびに更新されません。

```
filename student 'external-file-specification';
```

```
libname myv9lib 'SAS-library';  7

proc print data=myv9lib.class;
run;  8

proc print data=myv9lib.problems;
   format date datetime18.;
run;  9
```

1 ライブラリ MyV9Lib を参照します。ファイル参照名 Student を割り当てるファイルが格納されている場所を指定します。

2 データファイル Problems と、SAS ビュー Class を作成し、両方のデータセットの列名を指定します。

3 ファイル参照名 Student で参照されるファイルを選択し、ファイル内の指定位置にある文字形式のデータを選択します。列名を割り当てます。

4 列 Name、Major または Credits のデータ値がブランク(または欠損値)の場合は、欠損値が見つかったオブザベーションに 01 というコードを割り当てます。また、エラーに SAS System の日時コードを割り当て、それらの情報をデータファイル Problems に書き出します。

5 credits の数値が 0 より大きく 90 より小さい場合は、オブザベーションにコード 02 を割り当ててデータファイル Problems に格納し、そのオブザベーションに SAS System の日時コードを割り当てます。

6 指定したエラーがまったくないその他のオブザベーションはすべて、SAS ビュー MyV9Lib.Class に配置します。

7 FILENAME ステートメントを使用して、外部ファイルにファイル参照名 Student を割り当てます。LIBNAME ステートメントを使用して、SAS データライブラリにライブラリ参照名 MyV9Lib を割り当てます。

8 最初の PRINT プロシジャは、SAS ビュー MyV9Lib.CLASS を呼び出します。このビューは、ファイル参照名 Student で参照されるファイルから、その時点でのデータを抽出します。

9 この PRINT プロシジャは、データファイル MyV9Lib.Problems の内容を出力します。

PROC SQL ビュー

PROC SQL ビューとは、名前を付けて保存した、再利用可能な PROC SQL クエリ式のことです。PROC SQL ビューを使用すると、FROM 句に指定されているデータセット(テーブル)または SAS ビューからデータを取得できます。PROC SQL ビューがアクセスするデータは、データセットまたは SAS ビューにあるサブセット化したデータです。

PROC SQL ビューでデータを読み書きできる対象は次のとおりです。

- DATA ステップビュー
- SAS データファイル
- 他の PROC SQL ビュー
- SAS/ACCESS ビュー
- DB2 データや Oracle データなどの DBMS データ

PROC SQL ビューの作成方法と使用方法の詳細については、“SQL” (*SAS SQL Procedure User's Guide*)を参照してください。

以前のバージョンで作成した PROC SQL ビューの使用法については、33 章, “SAS 9.4 における、以前のリリースの SAS ファイルとの互換性” (717 ページ)を参照してください。

DATA ステップビューと PROC SQL ビューの比較

DATA ステップビューと PROC SQL ビューのどちらを使用するかを決定するには、各 SAS ビューの特性を把握しておく必要があります。

- DATA ステップビュー
  - DATA ステップビューは、DO ループや IF-THEN-ELSE ステートメントなどの DATA ステップ処理の機能を利用できるため、汎用性に優れています。
  - DATA ステップビューには、更新機能がありません。すなわち、DATA ステップビューは、それがアクセスするデータを直接変更することはできません。
  - DATA ステップビューのデータについては、使用前に処理対象を限定することはできません。このため、SAS ビューの一部のデータだけが必要な場合であっても、DATA ステップビュー全体をメモリにロードし、不要な部分をそこからすべて削除する必要があります。
  - WHERE 句を DATA ステップビューに適用する場合、WHERE 句は DATA ステップビューエンジンが評価します。
- PROC SQL ビュー
  - PROC SQL ビューでは、さまざまなファイル形式のデータを結合できます。
  - PROC SQL ビューでは、参照するデータの読み取りと更新を実行できます。
  - PROC SQL ビューでは、DATA ステップ処理で使用できる WHERE 句よりも多くの種類の WHERE 句を使用できます。また、CONNECT TO 句を使用できるため、パススルー機能を利用して、DBMS に対して簡単に SQL ステートメントを送信したり、データを渡したりできます。
  - SQL 言語の機能を利用して、処理前にデータをサブセット化することもできます。これにより、大きな SAS ビューを取り扱うときのメモリ消費が少なくなります。ただし、ビューに含まれるデータのごく一部のみを選択する必要があります。
  - PROC SQL ビューでは、DATA ステッププログラムを使用しません。
  - WHERE 句が PROC SQL ビューに適用される際、その WHERE 句の評価は、PROC SQL ビューエンジンが行う場合、またはライブラリのエンジンが行う場合があります。

SAS/ACCESS ビュー

SAS/ACCESS ビューは、インターフェイスビューであり、ビューディスクリプタとも呼ばれます。このビューは、対応するアクセスディスクリプタで定義されている DBMS データにアクセスします。

SAS/ACCESS ソフトウェアを使用すると、アクセスディスクリプタおよびビューディスクリプタを作成して、DBMS テーブルまたは DBMS ビューの一部あるいは全部を定義し、それらにアクセスすることができます。また、ビューディスクリプタを使用して DBMS データを更新することもできますが、いくつかの制約があります。

SAS/ACCESS ソフトウェアの中には、動的 LIBNAME Engine インターフェイスを実装しているものがあります。SAS/ACCESS ソフトウェアの LIBNAME ステートメントを使用できる場合は、それを使用して、DBMS データに SAS ライブラリ参照名を割り当てることをお勧めします。このステートメントは、アクセスディスクリプタやビューディスクリプタよりも効率がよく、簡単に使用できます。SAS/ACCESS ソフトウェアの動的 LIBNAME Engine を利用して DBMS に SAS ライブラリ参照名を割り当てると、DBMS のデータを SAS データと同じように処理できるようになります。SAS/ACCESS ソフトウェアの動的 LIBNAME Engine を利用して DBMS に SAS ライブラリ参照名を割り当てると、DBMS のデータを SAS データと同じように処理できるようになります。つまり、ビューディスクリプタのかわりに、ネイティブビューである DATA ステップビューや PROC SQL ビューを使用して、DBMS データにアクセスすることができます。

SAS/ACCESS ソフトウェアの機能の詳細については、31 章, “SAS/ACCESS ソフトウェアについて” (699 ページ) か、または使用しているデータベースに対応した SAS/ACCESS ソフトウェアのドキュメントを参照してください。

以前のバージョンで作成した SAS/ACCESS ビューディスクリプタの使用法については、33 章, “SAS 9.4 における、以前のリリースの SAS ファイルとの互換性” (717 ページ)を参照してください。

注: SAS 9 以降は、リレーショナル DBMS データにアクセスする場合、PROC SQL ビューの使用を推奨します。CV2VIEW プロシジャを使用して、既存の SAS/ACCESS ビューディスクリプタを PROC SQL ビューに変換できます。これにより、LIBNAME ステートメントを使用してデータにアクセスできます。詳細については、*SAS/ACCESS for Relational Databases: Reference* にある CV2VIEW プロシジャの説明を参照してください。

28 章

コンパイル済みストアド DATA ステッププログラム

コンパイル済みストアド DATA ステッププログラムの定義

コンパイル済みストアド DATA ステッププログラムとは、コンパイル後 SAS ライブラリに格納された DATA ステッププログラムが含まれている SAS ファイルのことです。コンパイル済みストアドプログラムは、すでにコンパイル済みの中間コードを含んでいるので、必要なときに再コンパイルすることなく直接実行することができます。コンパイル済みストアド DATA ステッププログラムは、メンバタイプ PROGRAM を持つ SAS ファイルです。

注: コンパイル済みストアドプログラムとして使用できるのは、DATA ステップのアプリケーションに限られます。コンパイル済みストアドプログラムには、グローバルステートメント以外の SAS 言語要素を保存できます。ソースプログラム中にグローバルステートメントを記述した場合、コンパイル済みプログラムは保存されますが、グローバルステートメントは保存されません。また、グローバルステートメントに関しては、SAS ログに警告メッセージは表示されません。

コンパイル済みストアド DATA ステッププログラムの使用

コンパイル済みストアド DATA ステッププログラムは、主にプロダクションジョブを実行する場合に使用します。この DATA ステッププログラムはすでにコンパイル済みの中間コードを含んでいるので、コンパイルのために毎回リソースを消費するということがなく、必要に応じてプログラムを直接実行できます。DATA ステップに多くのステートメントが記述されている場合は、節約できるリソース量も大きくなります。SAS System の新しいバージョンをインストールする場合にも、ソースコードを再コンパイルする必要はありません。

コンパイル済みストアド DATA ステッププログラムに関する制限事項と必要条件

コンパイル済みストアド DATA ステッププログラムを使用する場合、次のような制限事項と必要条件が適用されます。

- コンパイル済みストアド DATA ステッププログラムとして使用できるのは、DATA ステップアプリケーションに限られます。
- コンパイル済みストアド DATA ステッププログラムの中に、グローバルステートメントを記述することはできません。ソースプログラムの中に、FILENAME、FOOTNOTE、LIBNAME、OPTIONS、TITLE などのグローバルステートメントを記述した場合、コンパイル済みプログラムは保存されますが、グローバルステートメントは保存されません。SAS ログには警告メッセージは表示されません。
- 生データは、コンパイル済みプログラムには格納できません。

動作環境の情報
: 互換性のない動作環境には、コンパイル済みプログラムを移動できません。そのかわりに、ソースコードを再コンパイルし、新しいコンパイル済みプログラムを格納する必要があります。

ただし、コンパイル済みプログラムを、互換性のある別の動作環境に移動することはできます。

コンパイル済みストアド DATA ステッププログラムの処理の仕組み

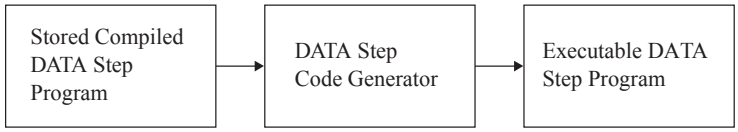
まず、SAS ソースプログラムをコンパイルし、コンパイル済みコードを格納します。次に、コンパイル済みのプログラムを実行し、必要であれば、入力データセットと出力データセットをリダイレクトします。

コンパイル時に DATA ステップを処理し、SAS ファイルの中に、プログラムおよび関連するデータテーブルの中間コードを格納します。この中間コードは、ストアドプログラムを実行したときに処理されます。次の図は、コンパイル済みストアド DATA ステッププログラムの作成プロセスを示しています。

図 28.1 コンパイル済みストアドプログラムの作成

ストアドプログラムが実行されるときに、コンパイラの作成した中間コードが展開され、その動作環境で実行可能な機械語コードが生成されます。次の図は、コンパイル済みストアド DATA ステッププログラムの実行プロセスを示しています。

図 28.2 コンパイル済みストアドプログラムの実行



ストアドプログラムの移送、コピー、名前の変更、削除を行うには、DATASETS プロシジャ、ウィンドウ環境のユーティリティウィンドウを使用します。

コンパイル済みストアド DATA ステッププログラムの作成

コンパイル済みストアド DATA ステッププログラムの作成の構文

コンパイル済みストアド DATA ステッププログラムを作成するための構文は、次のとおりです。

DATA *data-set-name(s)* / PGM=*stored-program-name*
<(<*password-option*><SOURCE=*source-option*>)>;

各引数の意味は次のとおりです。

data-set-name
DATA ステップのソースプログラムによって作成される出力データセットとして有効な SAS 名を指定します。1 レベル名または 2 レベル名を指定できます。DATA ステートメントに、データセット名は複数指定できます。

stored-program-name
ストアドプログラムを含む有効な SAS ファイル名を指定します。名前には 1 レベル名も指定できますが、通常は 2 レベル名を指定します。SAS データライブラリに保存されたストアドプログラムには、メンバタイプ PROGRAM が割り当てられます。

password-option
コンパイル済みストアド DATA ステッププログラムにパスワードを割り当てます。

source-option
ソースコードを保存または暗号化します。

DATA ステートメントの完全な説明については、*SAS Statements: Reference* を参照してください。

DATA ステッププログラムのコンパイルと格納のプロセス

DATA ステッププログラムをコンパイルして格納するには、次の操作を行います。

1. DATA ステッププログラムを記述、テスト、デバッグします。

   生データファイルを読み込む場合や生データを外部ファイルに出力する場合、INFILE ステートメントと FILE ステートメントの中では、実際のファイル名ではなくファイル参照名を使用するように指定します。この指定により、コンパイル済みストアドプログラムの実行時に入力または出力データセットをリダイレクトできます。

2. プログラムが正しく動作したら、DATA ステートメントで PGM=オプションを使用してプログラムをサブミットします。

   PGM=オプションは、プログラムをコンパイルし、実行せずに、このオプションで指定した SAS ファイルにコンパイル済みコードを保存するように指定します。プログラムが保存されると、SAS ログにメッセージが表示されます。

注: デフォルトでは SOURCE=SAVE オプションまたは SOURCE=ENCRYPT オプションが有効となるため、ソースコードは自動的に保存されます。

注: アプリケーションを別の動作環境に移送する場合は、ソースコードを移送先の環境で再コンパイルし、新しいコンパイル済みプログラムを作成する必要があります。

例:コンパイル済みストアド DATA ステッププログラムの作成

次の例では、入力 SAS データセット In.Sample の情報を使用し、植物のコードに基づいて植物の種類を割り当てます。グローバルステートメントである LIBNAME ステートメントは、ファイルの保存場所を識別するために必要ですが、SAS が格納した DATA ステップ Stored.Sample の中には記述されていないことに注意が必要です。

```
libname in 'SAS-library';
libname stored 'SAS-library';

data out.sample / pgm=stored.sample;
   set in.sample;
   if code = 1 then
      do;
         Type='Perennial';
         number+4;
      end;
   else
   if code = 2 then
      do;
         Type='Annual';
         number+10;
      end;
   else
      do;
         Type='ERROR';
         Number=0;
      end;
run;
```

ログ 28.1 *ストアド DATA ステッププログラムを示す SAS ログの一部*

```
...NOTE:DATA STEP program saved on file Stored.Sample.NOTE:A stored DATA STEP
program cannot run under a different operating system.NOTE:DATA statement used
(Total process time): real time           0.17 seconds cpu time            0.01
seconds
```

コンパイル済みストアド DATA ステッププログラムの実行

コンパイル済みストアド DATA ステッププログラムの実行の構文

コンパイル済みストアド DATA ステッププログラムを実行するための構文は、次のとおりです。これにより、ソースコードを取得したり、入力または出力データセットをリダイレクトしたりできます。

…global SAS statements…

DATA PGM=*stored-program-name* <(*password-option*)>;

<DESCRIBE;>

<REDIRECT INPUT | OUTPUT *old-name-1 = new-name-1<...old-name-n = new-name-n*>;>

<EXECUTE;>

global SAS statements
プログラムの実行時に必要になるグローバルステートメントを指定します。このようなステートメントの例としては、入力先や出力先を示す FILENAME ステートメントや LIBNAME ステートメントなどがあります。

stored-program-name
ストアドプログラムを含む有効な SAS ファイル名を指定します。1 レベル名または 2 レベル名を指定できます。

password-option
コンパイル済みストアド DATA ステッププログラムにアクセスするためのパスワードを指定します。

DESCRIBE
コンパイル済みストアド DATA ステッププログラムまたは DATA ステップビューのソースコードを SAS ログに表示します。

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。プログラムに複数のパスワードがかけられている場合、最も制限が強いパスワードを指定する必要があります。最も制限が強いパスワードが変更パスワードで、最も制限が弱いパスワードが読み取りパスワードです。詳細については、“DESCRIBE Statement” (*SAS Statements: Reference*)を参照してください。

INPUT | OUTPUT
入力データセットや出力データセットをリダイレクトするかどうかを指定します。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。

EXECUTE
コンパイル済みストアド DATA ステッププログラムを実行します。

DATA ステートメントの完全な説明については、“DATA Statement” (*SAS Statements: Reference*)を参照してください。

コンパイル済みストアド DATA ステッププログラムの実行プロセス

コンパイル済みストアド DATA ステッププログラムを実行するには、次の操作を実行します。

1. 実行するストアドプログラムごとに、DATA ステップを記述します。この DATA ステップでは、DATA ステートメントの PGM=オプションでストアドプログラム名を指定し、必要に応じてパスワードも指定します。次のタスクのいずれも実行できます。
   - 記述した DATA ステップを別プログラムとしてサブミットできます。
   - 記述した DATA ステップを、他の DATA ステップおよびプロシジャ(PROC)ステップを格納できる大きな SAS プログラムに含めることができます。
   - REDIRECT ステートメントを使用することで、ストアドプログラムを実行するたびに、異なる入力データセットおよび出力データセットを割り当てられます。
2. DATA ステップをサブミットします。各 DATA ステップは、RUN ステートメントまたはその他のステップ境界で終了させてください。

グローバルステートメントの使用

コンパイル済みストアド DATA ステッププログラムを作成または実行する際に、FILENAME や LIBNAME などのグローバル SAS ステートメントを使用できます。ただし、DATA ステッププログラムをコンパイルして保存するために使用するグローバルステートメントは、DATA ステッププログラムと一緒には保存できません。

出力のリダイレクト

ファイル参照名を使用すると、外部ファイルをリダイレクトできます。入力データセットや出力データセット名を変更するには、REDIRECT ステートメントを使用します。

REDIRECT ステートメントを使用すると、入力データや出力データを指定のデータセットにリダイレクトできます。REDIRECT ステートメントを使用できるのは、コンパイル済みストアド DATA ステッププログラムに対してのみです。

注: 外部ファイルに格納されている入力データや出力データをリダイレクトするには、FILENAME ステートメントを記述して、ソースプログラム内のファイル参照名を特定の外部ファイルに割り当てます。

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が持つ変数の数と属性は、ソースコード内の SET、MERGE、MODIFY、UPDATE ステートメントに記述した入力データセットにおける変数と一致している必要があります。一致していない場合は、次のような結果になります。

- 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。
- リダイレクトされるデータセットに余分な変数が存在する場合、ストアドプログラムの処理は停止し、SAS ログにエラーメッセージが出力されます。
- 変数の種類属性が異なる場合は、ストアドプログラムの処理は停止し、SAS ログにエラーメッセージが出力されます。

コンパイル済みストアド DATA ステッププログラムのソースコードの出力

SAS では、コンパイル済みストアド DATA ステッププログラムを実行するときに、DESCRIBE ステートメントと EXECUTE ステートメントを一緒に使用すると、ログにソースコードが書き込まれます。次の例では、コンパイル済みストアド DATA ステッププログラムを実行しています。DESCRIBE ステートメントを指定することで、ソースコードを SAS ログに表示します。

```
data pgm=stored.sample;
   describe;
   execute;
run;
```

ログ 28.2 DESCRIBE ステートメントによって生成されたソースコードを示す SAS ログの一部

```
...190  data pgm=stored.sample; 191     describe; 192     execute; 193  run;
NOTE:DATA step stored program Stored.Sample is defined as:data out.sample /
pgm=stored.sample; set in.sample; if code = 1 then do; Type='Perennial'; number
+4; end; else if code = 2 then do; Type='Annual'; number+10; end; else do;
Type='ERROR'; Number=0; end; run; NOTE:DATA STEP program loaded from file
Stored.Sample.NOTE:There were 7 observations read from the data set
In.Sample.NOTE:The data set OUT.SAMPLE has 7 observations and 4
variables.NOTE:DATA statement used (Total process time): real time 0.03 seconds
cpu time 0.00 seconds
```

DESCRIBE ステートメントに関する詳細については、*SAS Statements: Reference* を参照してください。

例:コンパイル済みストアド DATA ステッププログラムの実行

次の DATA ステップは、“例:コンパイル済みストアド DATA ステッププログラムの作成” (678 ページ)で作成したストアドプログラム Stored.Sample 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 を指定しています。このプログラムを実行して得られた出力結果は、データセット Totals.Sample へとリダイレクトされて格納されます。ログ 28.3 (681 ページ) は、SAS ログの一部を示しています。

```
libname in 'SAS-library';
libname base 'SAS-library';
libname totals 'SAS-library';
libname stored 'SAS-library';

data pgm=stored.sample;
   redirect input in.sample=base.sample;
   redirect output out.sample=totals.sample;
run;
```

ログ 28.3 リダイレクト先の出力ファイルを示す SAS ログの一部

```
...224  data pgm=stored.sample; 225     redirect input in.sample=base.sample;
226     redirect output out.sample=totals.sample; 227  run; NOTE:DATA STEP
program loaded from file Stored.Sample.NOTE:There were 7 observations read from
the data set BASE.SAMPLE.NOTE:The data set Totals.Sample has 7 observations and
4 variables.NOTE:DATA statement used (Total process time): real time 0.12
seconds cpu time 0.01 seconds 228 proc printto; run;
```

コンパイル済みストアド DATA ステッププログラムと DATA ステップビューの相違点

コンパイル済みストアド DATA ステッププログラムと、DATA ステップビューは、どちらもほぼ同じ機能を備えています。両者とも、他のファイルに保持されているデータの取得や処理を行う DATA ステッププログラムを格納します。両者には同じ制限事項と必要条件が適用されます(“コンパイル済みストアド DATA ステッププログラムに関する制限事項と必要条件” (676 ページ)を参照)。DATA ステップビューの詳細については、“DATA ステップビュー” (668 ページ)を参照してください。

コンパイル済みストアド DATA ステッププログラムと DATA ステップビューでは、次の点が異なります。

- コンパイル済みストアド DATA ステッププログラムは、DATA ステートメントの PGM=オプションで指定されたとき、明示的に実行されます。コンパイル済みストアド DATA ステップは、主に定型的な処理で使用します。
- DATA ステップビューは、他の DATA ステップまたはプロシジャ(PROC)ステップによって入力データセットとして参照されたときに、暗黙的に実行されます。主に、呼び出すプロシジャや DATA ステップにデータを一度に 1 オブザベーションずつ引き渡す用途で使用されます。
- コンパイル済みストアド DATA ステッププログラムの実行時には、REDIRECT ステートメントを使用できます。DATA ステップビューを使用する場合は、REDIRECT ステートメントを使用できません。

DATA ステッププログラムの例

品質管理アプリケーション

この例では、コンパイル済みストアド DATA ステッププログラムを単純な品質管理アプリケーションで使用する方法を示します。このアプリケーションでは、生データファイルを処理します。ソースプログラムでは、INFILE ステートメントでファイル参照名 Daily を使用しています。ストアドプログラムを実行するための各 DATA ステップに FILENAME ステートメントを記述すると、ファイル参照名 Daily を別の外部ファイルに割り当てることができます。

次のステートメントで、プログラムをコンパイルし、格納します。

```
libname stored 'SAS-library-1';

data flaws / pgm=stored.flaws;
   length Station $ 15;
   infile daily;
   input Station $ Shift $ Employee $ Flaws;
   Total + Flaws;
run;
```

次のプログラムは、コンパイル済みストアドプログラムを実行し、出力データセットをリダイレクトし、実行結果を出力しています。

```
libname stored 'SAS-library-1';
libname testlib 'SAS-library-2';

data pgm=stored.flaws;
   redirect output flaws=testlib.daily;
run;

proc print data=testlib.daily;
   title 'Quality Control Report';
run;
```

アウトプット 28.1 *品質管理アプリケーションの出力*

Quality Control Report

Obs	Station	Shift	Employee	Flaws	Total
1	cambridge	1	Lin	3	3
2	Northampton	1	Kay	0	3
3	Springfield	2	Sam	9	12

コンパイル済みストアド DATA ステッププログラムを実行したり、実行結果を出力したりするときは、TITLE ステートメントを使用できます。

29 章

DICTIONARY テーブル

DICTIONARY テーブルの定義

DICTIONARY テーブルとは、現在の SAS セッションで使用中か使用可能な SAS ライブラリ、SAS データセット、SAS マクロ、外部ファイルに関する情報が格納されている、読み取り専用の SAS ビューです。DICTIONARY テーブルには、現在有効な SAS システムオプションの設定も含まれています。

ユーザーが DICTIONARY テーブルにアクセスすると、SAS System は SAS セッションの現在の状態を判定した後、状況に応じて必要な情報を返します。この処理は、DICTIONARY テーブルへのアクセスが行われるたびに実行されるため、ユーザーは常に現在の情報を得ることができます。

SAS プログラムから DICTIONARY テーブルにアクセスするには、次の方法のいずれかを使用します。

- ライブラリ参照名 DICTIONARY を使用して、同テーブルに対する PROC SQL クエリを実行します。
- SAS プロシジャまたは DATA ステップを使用して、Sashelp ライブラリ内の同テーブルの PROC SQL ビューを参照します。

DICTIONARY テーブル、利用可能な DICTIONARY テーブルのリスト、およびそれらに関連付けられている Sashelp ビューの詳細については、*SAS SQL プロシジャユーザーガイド*を参照してください。

DICTIONARY テーブルを表示する方法

DICTIONARY テーブルについて

DATA ステップや SAS プロシジャで DICTIONARY テーブルを実際に使用する前に、テーブルの内容を表示して、現在の SAS セッションに関する情報を確認できます。

状況によっては、DICTIONARY テーブルのサイズがかなり大きくなることがあります。このような場合には、自分が興味のあるデータのみを含む DICTIONARY テーブルの一部を表示できます。DICTIONARY テーブルの一部だけを表示する最適な方法は、SQL プロシジャで WHERE 句を使用することです。

DICTIONARY テーブルを表示する方法

各 DICTIONARY テーブルには、Sashelp ライブラリ内に関連付けられている PROC SQL ビューがあります。VIEWTABLE または FSVIEW ユーティリティを使用して対応する Sashelp ビューを開くことにより、DICTIONARY テーブルのすべての内容を表示できます。この方法により表示される内容は、DESCRIBE TABLE ステートメントの出力結果(“DICTIONARY テーブルの要約を表示する方法” (686 ページ)を参照)よりも詳細です。

ウィンドウ環境で、VIEWTABLE または FSVIEW ユーティリティを使用して DICTIONARY テーブルを表示する方法を下記に示します。

1. SAS セッションで**エクスプローラ**ウィンドウを呼び出します。
2. Sashelp ライブラリを選択します。Sashelp ライブラリのメンバの一覧が表示されます。
3. V で始まる名前(例: VMEMBER など)を持つ SAS ビューを選択します。

   対応するビューの内容を含む **VIEWTABLE** ウィンドウが表示されます。z/OS の場合、表示したいメンバのコマンドフィールドに文字'O'をタイプした後、Enter キーを押します。対応するビューの内容を含む **FSVIEW** ウィンドウが表示されます。

VIEWTABLE ウィンドウに表示される列の見出しは、ラベルです。列見出しの実際の名前を表示するには、**表示** ⇨ **列名**を選択します。

DICTIONARY テーブルの要約を表示する方法

PROC SQL の中で DESCRIBE TABLE ステートメントを使用すると、DICTIONARY テーブルの内容の要約が作成されます。ここでは、DESCRIBE TABLE ステートメントを使用して、DICTIONARY.INDEXES テーブルの要約ファイルを作成する例を示します。(このテーブルの Sashelp ビューは、Sashelp.VINDEX になります)。

```
proc sql;
  describe table dictionary.indexes;
```

この DESCRIBE TABLE ステートメントの実行結果は、次の SAS ログのとおりです。

```
NOTE: SQL table DICTIONARY.INDEXES was created like:

create table DICTIONARY.INDEXES
  (
   libname char(8) label='Library Name',
```

```
  memname char(32) label='Member Name',
  memtype char(8) label='Member Type',
  name char(32) label='Column Name',
  idxusage char(9) label='Column Index Type',
  indxname char(32) label='Index Name',
  indxpos num label='Position of Column in Concatenated Key',
  nomiss char(3) label='Nomiss Option',
  unique char(3) label='Unique Option
 );
```

- 各行の先頭の語は、列(変数)名です。列(変数)を参照する SAS ステートメントを記述するときに、この名前を使用する必要があります。
- 列名の後にある語は、変数の種類と列の幅の指定です。
- **label=**に続く名前は、列(変数)のラベルです。

テーブルの内容が確認できたら、PROC SQL ステップで WHERE 句を利用することで、SAS ビューの一部を抽出できます。

DICTIONARY テーブルのサブセットの表示方法

大きな DICTIONARY テーブルを処理する場合、そのテーブルの一部のみが必要なときは、PROC SQL で WHERE 句を使用して、元のテーブルのサブセットを抽出します。次の PROC SQL ステートメントでは、WHERE 句を使用して DICTIONARY.INDEXES からデータを抽出しています。

```
proc sql;
  title 'Subset of the DICTIONARY.INDEX View';
  title2 'Rows with Column Name equal to STATE';
  select libname, memname, name
    from dictionary.indexes
    where name = 'STATE';
quit;
```

出力結果は次のようになります。

アウトプット 29.1 *SQL プロシジャによりサブセット化した WHERE ステートメントの出力*

Subset of the DICTIONARY.INDEX View
Rows with Column Name equal to STATE

Library Name	Member Name	Column Name
SASHELP	PLFIPS	STATE
MAPS	USAAC	STATE
MAPS	USAAC	STATE
MAPS	USAAS	STATE
MAPSSAS	USAAC	STATE
MAPSSAS	USAAC	STATE
MAPSSAS	USAAS	STATE

DICTIONARY テーブル内の多くの文字値は、すべて大文字で格納されていることに注意してください。このことを念頭に置いてクエリを設計する必要があります。

注意:

CONTENTS OUT=データセットオプションに指定されている GENNUM 変数の値を、DICTIONARY テーブル内の GEN 変数の値と混同しないでください。 CONTENTS プロシジャまたはステートメント内にある GENNUM 変数の値は、データセットの特定の世代を意味します。一方、DICTIONARY テーブル内にある GEN 変数の値は、データセットの総世代数を意味します。

DICTIONARY テーブルとパフォーマンス

DICTIONARY テーブルに対してクエリを行うと、そのテーブルに関する情報を収集できます。クエリの対象となる DICTIONARY テーブルによっては、このプロセスにライブラリの検索、テーブルのオープン、SAS ビューの実行が含まれている場合があります。その他の SAS プロシジャや DATA ステップと異なり、PROC SQL では、選択処理を開始する前にクエリを最適化することにより、このプロセスのパフォーマンスを改善できます。このため、多くの場合、DICTIONARY テーブルの情報にアクセスするには、SAS プロシジャや DATA ステップで Sashelp ビューを使用するよりも、PROC SQL を使用した方がより効率的になります。

たとえば、次の 2 つのプログラムは同じ結果を生成しますが、PROC SQL ステップの方がより高速になります。これは、Sashelp.VCOLUMN ビューにより使用されるテーブルを開く前に WHERE 句が処理されるためです。

```
data mytable;
   set sashelp.vcolumn;
   where libname='WORK' and memname='SALES';
run;

proc sql;
   create table mytable as
      select * from sashelp.vcolumn
```

```
      where libname='WORK' and memname='SALES';
quit;
```

注: SAS は、DICTIONARY テーブルの情報をクエリ間で維持しません。DICTIONARY テーブルのクエリを行うたびに、新しい検索プロセスが開始されます。

同じ DICTIONARY テーブルのクエリを何度も行う場合、必要な情報を含む一時 SAS データセットを作成し、そのデータセットに対してクエリを実行することにより、さらにパフォーマンスを高速化できます。一次データセットを作成するには、DATA ステップの SET ステートメントか、SQL プロシジャの CREATE TABLE AS ステートメントを使用します。

30 章

SAS カタログ

SAS カタログの定義

SAS カタログとは、SAS System におけるさまざまな種類の情報を、カタログエントリと呼ばれる小さな単位で格納する特殊な SAS ファイルです。個々のエントリには、その目的を示すエントリタイプが指定されています。単一の SAS カタログには、複数の種類のカタログエントリを含めることができます。カタログエントリには、キー定義などのシステム情報を含むものがあります。それ以外に、ウィンドウ定義、ヘルプウィンドウ、出力形式、入力形式、マクロ、グラフィック出力などのアプリケーション情報を含むカタログエントリもあります。SAS エクスプローラや CATALOG プロシジャなど、SAS System のさまざまな機能を使用すると、カタログの内容を一覧表示できます。

SAS カタログ名

カタログ名の構成要素

SAS カタログエントリは、次の形式の最大 4 レベル名で表されます。

libref.catalog.entry-name.entry-type

カタログ全体を指定する場合、通常、次のような 2 レベル名を使用します。

libref.catalog

libref
カタログが属している SAS ライブラリの論理名(ライブラリ参照名)を指定します。

catalog
カタログファイルの有効な SAS 名を指定します。

エントリ名とエントリタイプは、一部の SAS プロシジャで必要になります。エントリタイプが、別の場所で指定されている場合やコンテキストから確定できる場合は、エントリ名を単独で使用できます。エントリ名とエントリタイプを指定するには、次の形式を使用します。

entry-name.entry-type

entry-name
カタログエントリの有効な SAS 名を指定します。

entry-type
エントリの作成時に SAS System によって割り当てられるエントリタイプを指定します。

カタログ内にある情報へのアクセス

Base SAS ソフトウェアでは、通常、SAS カタログエントリに格納されている情報が処理で必要になると、SAS System がカタログエントリに自動的にアクセスします。その他の SAS ソフトウェアプロダクトでは、ユーザーがさまざまなプロシジャでカタログエントリを指定する必要があります。SAS プロシジャや SAS ソフトウェアプロダクトによって必要条件が異なるため、詳細については、該当するプロシジャや SAS ソフトウェアプロダクトのドキュメントを参照してください。

SAS カタログの管理ツール

SAS System には、カタログ内にあるエントリを管理するためのいくつかのツールが用意されています。Base SAS ソフトウェアにおけるツールは、CATALOG プロシジャと CEXIST 関数です。その他の管理ツールとしては、SAS エクスプローラがあります。SAS エクスプローラを使用すると、SAS カタログの内容を表示できます。多くの対話型ウィンドウプロシジャには、エントリを管理するためのカタログディレクトリウィンドウが含まれています。カタログ管理に使用できるツールには次のものがあります。

CATALOG プロシジャ
DATASETS プロシジャと同等のプロシジャです。CATALOG プロシジャを使用して、カタログ内のエントリのコピー、削除、一覧表示、名前変更を行います。

CEXIST 関数
SAS カタログまたはカタログエントリの存在を確認します。詳細については、*SAS Functions and CALL Routines: Reference* にある CEXIST 関数の説明を参照してください。

CATALOG ウィンドウ
対話型ウィンドウ環境でいつでも表示できるウィンドウです。このウィンドウには、指定したカタログの各エントリについてのエントリ名、エントリタイプ、説明、最終更新日付が表示されます。**CATALOG** ウィンドウのコマンドを使用すると、カタログエントリを編集できます。SAS エクスプローラでカタログファイルをダブルクリックして、カタログエントリを表示および編集することもできます。

カタログディレクトリウィンドウ
SAS/AF ソフトウェア、SAS/FSP ソフトウェア、SAS/GRAPH ソフトウェアの一部のプロシジャで利用できるウィンドウです。カタログディレクトリウィンドウには、エントリ名、エントリタイプ、説明、最終更新日付など、**CATALOG** ウィンドウに表示される情報と同じ種類の情報が一覧表示されます。対話型ウィンドウプロシジャのカタログディレクトリウィンドウの詳細については、各プロシジャの説明を参照してください。

プロファイルカタログ

プロファイルカタログの定義

プロファイルカタログ(Sasuser.Profile)とは、SAS System での作業方法をカスタマイズするためのカタログです。SAS はこのカタログには、ファンクションキーの定義、グラフィックアプリケーション用フォント、ウィンドウの属性および対話型のウィンドウプロシジャからのその他の情報を保存します。

プロファイルカタログの利用

Sasuser.Profile カタログの情報は、ユーザーが処理で必要とするときに自動的にアクセスされます。たとえば、**KEYS** ウィンドウを表示し、設定を変更するたびに、KEYS エントリタイプの新しい設定が格納されます。同様に、対話型ウィンドウプロシジャの属性を変更して保存すると、適切なエントリ名およびエントリタイプの変更内容が格納されます。このウィンドウまたはプロシジャを使用すると、プロファイルカタログの情報が検索されます。

Sasuser.Profile の作成方法

SAS System は、初めてプロファイルカタログを参照しようとした場合に、それが存在しないことを検出すると、プロファイルカタログを作成します。対話型ウィンドウ環境を使用している場合は、最初の SAS セッションでのシステムの初期化中にこの動作が行われます。その他の実行モードを使用している場合は、プロファイルカタログを要求する SAS プロシジャを初めて実行したときに、プロファイルカタログが作成されます。

SAS System はその起動時に、既存の破損していない Sasuser.Profile カタログをチェックします。このカタログが検出されると、SAS System は Sasuser.Profile カタログを Sasuser.Profbak にコピーします。Sasuser.Profile カタログが破損した場合は、このバックアップが使用されます。詳細については、“プロファイルカタログがロックされた場合や破損した場合の修復方法” (694 ページ)を参照してください。

動作環境の情報

SASUSER ライブラリの実装形式は、動作環境によって異なります。SASUSER ライブラリの作成方法の詳細については、使用しているホストシステムに対応する SAS ドキュメントを参照してください。

デフォルト設定

SAS セッションのデフォルト設定は、Sashelp インストールライブラリの各種のカタログに保存されます。キー設定やその他の設定を変更しない場合、デフォルト設定が使用されます。変更した場合は、新しい情報がカタログ Sasuser.Profile に保存されます。デフォルト設定を復元するには、CATALOG プロシジャまたは **CATALOG** ウィンドウを使用して、プロファイルカタログの該当するエントリを削除します。その後、デフォルトで、SASHELP ライブラリから対応するエントリが使用されます。

SAS セッション中に、ユーザーはウィンドウのサイズ変更や配置などのカスタマイズを行い、それを Sasuser.Profile に保存できます。

プロファイルカタログがロックされた場合や破損した場合の修復方法

Sasuser.Profile カタログは、ロックされる場合や破損する場合があります。カタログがロックされた場合や破損した場合、SAS System は Sasuser.Profile や Sasuser.Profbak を使用してカタログを置き換えます。

Sasuser.Profile カタログがロックされた場合、SAS System は Sashelp.Profile が存在するかどうか調べます。Sashelp.Profile が存在する場合、SAS System はそれを Work.Profile へコピーし、それ以降はカスタマイズ情報を Sasuser.Profile ではなく Work.Profile に保存します。この場合、SAS ログに次のメッセージが出力されます。

```
ERROR: Expecting page 1, got page -1 instead.
ERROR: Page validation error while reading SASUSER.PROFILE.CATALOG.
NOTE: Unable to open SASUSER.PROFILE. WORK.PROFILE will be opened instead.
NOTE: SASHELP.PROFILE has been copied to WORK.PROFILE.
NOTE: All profile changes will be lost at the end of the session.
```

Sasuser.Profile カタログが破損している場合、SAS System はその破損したカタログを Sasuser.Badpro へコピーします。その後、Sasuser.Profbak が存在するかどうかを調べます。Sasuser.Profbak が存在する場合、SAS System はそれを Sasuser.Profile へコピーします。この場合、以前のセッションで実施した Sasuser.Profile への変更はすべて失われます。この場合、SAS ログに次のメッセージが出力されます。

```
ERROR: Expecting page 1, got page -1 instead.
ERROR: Page validation error while reading SASUSER.PROFILE.CATALOG.
NOTE: A corrupt SASUSER.PROFILE has been detected. A PROFILE catalog can
      become corrupt when a SAS session is prematurely terminated.
NOTE: SASUSER.PROFILE.CATALOG has been renamed to SASUSER.BADPRO.CATALOG.
NOTE: SASUSER.PROFILE.CATALOG has been restored from SASUSER.PROFBAK.CATALOG.
NOTE: Changes made to SASUSER.PROFILE.CATALOG during the previous SAS session
      have been lost. The type of data stored in the PROFILE catalog is
      typically related to SAS session customizations such as key
      definitions, fonts for graphics, and window attributes.
```

Sasuser.Profile カタログが破損した場合に Sasuser.Profbak が存在しないならば、SAS System は Sashelp.Profile が存在するかどうかを調べます。Sashelp.Profile が存在する場合、SAS System はそれを Work.Profile へコピーし、それ以降はカスタマイズ情報を Sasuser.Profile ではなく Work.Profile に保存します。この場合、SAS ログに次のメッセージが出力されます。

```
ERROR: Expecting page 1, got page -1 instead.
ERROR: Page validation error while reading SASUSER.PROFILE.CATALOG.
```

```
NOTE: Unable to open SASUSER.PROFILE. WORK.PROFILE will be opened instead.
NOTE: SASHELP.PROFILE has been copied to WORK.PROFILE.
NOTE: All profile changes will be lost at the end of the session.
```

カタログ連結

定義

複数の SAS カタログを連結することで、それらを論理的に結合することができます。カタログを連結すると、1 つのカタログ名を使用して、複数のカタログの内容にアクセスすることができます。カタログの連結には、LIBNAME ステートメントを使用する連結と、CATNAME ステートメントを使用する連結の 2 種類があります。

LIBNAME カタログ連結
: LIBNAME ステートメントを利用してライブラリを連結することにより、カタログを連結します。複数のライブラリが論理的に結合されると、各ライブラリの同一名のカタログもすべて論理的に結合されます。

CATNAME カタログ連結
: グローバルな CATNAME ステートメントを利用してカタログを具体的に指定することにより、それらのカタログを連結します。CATNAME ステートメントによるカタログの連結では、メモリ内に論理カタログが作成されます。

例 1:LIBNAME ステートメントによるカタログの連結

次の LIBNAME ステートメントは、2 つの SAS ライブラリを連結します。

```
libname both ('SAS-library-1' 'SAS-library-2');
```

ライブラリ 1 のメンバ	ライブラリ 2 のメンバ
MyCat.CATALOG	MyCat.CATALOG
Table1.DATA	MyCat2.CATALOG
Table3.DATA	Table1.DATA
	Table1.INDEX
	Table2.DATA
	Table2.INDEX

連結ライブラリ参照名 Both には、次のメンバが含まれます。

連結ライブラリ参照名 Both
MyCat.CATALOG (ライブラリ 1 およびライブラリ 2 より)
MyCat2.CATALOG (ライブラリ 2 より)

連結ライブラリ参照名 Both
Table1.DATA (ライブラリ 1 より)
Table2.DATA (ライブラリ 2 より)
Table2.INDEX (ライブラリ 2 より)
Table3.DATA (ライブラリ 1 より)

ライブラリの連結では、Table1.INDEX は表示されませんが、Table2.INDEX は表示されます。インデックスに関連するデータファイルが連結ライブラリに含まれない場合、そのインデックスは表示されません。

ライブラリが連結された場合、連結されたカタログはメモリ内に論理的に存在しています。カタログ名の完全な名前は Both.MyCat.CATALOG です。このカタログは、*SAS-library-1* および *SAS-library-2* にそれぞれ存在する 2 つの物理カタログ MyCat.CATALOG を結合したものです。

連結カタログ Both.MyCat の内容を理解するには、まず、連結前の内容に注目します。元の 2 つの MyCat.CATALOG のカタログファイルに、次のエントリが含まれているとします。

ライブラリ 1 (MyCat.CATALOG)の内容	ライブラリ 2 (MyCat.CATALOG)の内容
A.FRAME	A.GRSEG
C.FRAME	B.FRAME
	C.FRAME

連結カタログ Both.MyCat には、次のエントリが含まれます。

Both.MyCat
A.GRSEG (ライブラリ 2 より)
A.FRAME (ライブラリ 1 より)
B.FRAME (ライブラリ 2 より)
C.FRAME (ライブラリ 1 より)

例 2:CATNAME ステートメントによるカタログの連結

CATNAME ステートメントの構文は、次のとおりです。

```
CATNAME libref.catref
          (libref-1.catalog-1 (ACCESS=READONLY)
           libref-n.catalog-n (ACCESS=READONLY));
```

カタログの連結を解除するための構文は、次のとおりです。

```
CATNAME libref.catref | _ALL_ CLEAR;
```

次の例では、CatDog という名前で定義されるライブラリ参照名を使用します。ライブラリ参照名 CatDog は、CATNAME ステートメントによる連結の定義範囲を確定します。

注: ライブラリ参照名 CatDog に Combined.CATALOG という名前のデータファイルがすでに存在する場合、このデータファイルは、CATNAME ステートメントにより連結されたカタログ CatDog.Combined が解除されるまでアクセスできなくなります。

ライブラリ 1 のメンバ	ライブラリ 2 のメンバ
MyCat.CATALOG	MyDog.CATALOG
Table1.DATA	MyCat2.CATALOG
Table3.DATA	Table1.DATA
	Table1.INDEX
	Table2.DATA
	Table2.INDEX

次の CATNAME ステートメントを実行するとします。

```
catname catdog.combined
          (library1.mycat (access=readonly)
           library2.mydog (access=readonly));
```

その場合、連結カタログ CatDog.Combined では、次のカタログが結合されます。

連結カタログ(CatDog.Combined)
MyCat.CATALOG (ライブラリ 1 より)
MyDog.CATALOG (ライブラリ 2 より)

注: CATNAME ステートメントによる連結で結合されるのは、名前を指定したカタログだけです。LIBNAME ステートメントによる連結では、複数のライブラリに同一名のカタログが存在する場合、それらのカタログは、ライブラリが連結されたときに結合されます。

前述の CATNAME ステートメントは、メモリ内に論理的に存在する、カタログ CATDOG.COMBINED.CATALOG を作成します。このカタログでは、ライブラリ 1 に存在する MyCat.CATALOG とライブラリ 2 に存在する MyDog.CATALOG という 2 つの物理カタログが結合されます。

連結カタログ Combined.CATALOG の内容を理解するには、まず、連結前の内容に注目します。元の 2 つのカタログファイルには、次のエントリが含まれています。

MyCat.CATALOG ライブラリ 1	MyDog.CATALOG ライブラリ 2
A.FRAME	A.GRSEG

MyCat.CATALOG ライブラリ 1	MyDog.CATALOG ライブラリ 2
C.FRAME	B.FRAME
	C.FRAME

連結カタログ Combined.CATALOG には、次のエントリが含まれます。

連結カタログ(Combined.CATALOG)の内容
A.GRSEG (MyDog より)
A.FRAME (MyCat より)
B.FRAME (MyDog より)
C.FRAME (MyCat より)

カタログ連結のルール

カタログの連結に関するルールは、連結に LIBNAME ステートメントと CATNAME ステートメントのどちらを使用するかには関係なく同じです。

- カタログエントリが入力または更新のために開かれる場合は、各エントリが検索され、指定されたエントリのうち最初に検出されたものが使用されます。
- カタログエントリが出力のために開かれる場合は、連結の最初に列挙されるカタログにファイルが作成されます。

  注: 新規のカタログエントリは最初のライブラリに作成されます。同一の名前のカタログエントリが連結の別の部分に存在するかどうかには関係しません。

  注: 更新のために開かれる連結の最初のカタログが存在しない場合、連結に存在する次のカタログにカタログエントリが書き込まれます。
- 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一覧を表示する場合は、同一のカタログエントリは常に 1 つしか表示されません。

  注: 連結したカタログの中にカタログエントリが複数回出現する場合でも、表示されるのは最初に検出されたカタログエントリのみです。

31 章

SAS/ACCESS ソフトウェアについて

SAS/ACCESS ソフトウェアの定義

SAS/ACCESS ソフトウェア
SAS/ACCESS ソフトウェアを利用すると、SAS System 以外のデータベース管理システム(DBMS)のデータや、一部の PC ファイル形式のデータを読み書きできます。SAS/ACCESS ソフトウェアでは、使用する DBMS に応じて、次のインターフェイス機能を 1 つ以上使用できます。

- 動的 LIBNAME Engine
- SQL パススルー機能
- ACCESS プロシジャおよびインターフェイスビューエンジン
- DBLOAD プロシジャ
- インターフェイス DATA ステップエンジン

本セクションでは、これらのインターフェイス機能について説明します。
SAS/ACCESS ソフトウェアでは、サポートする DBMS ごとに、これらのインターフェイス機能を 1 つ以上使用できます。SAS エンジンの詳細については、35 章, “SAS エンジン” (737 ページ) を参照してください。

注: DBMS データへアクセスするために SAS/ACCESS ソフトウェアの機能を使用するには、SAS/ACCESS ソフトウェアのライセンス契約が必要です。詳細については、各 DBMS 用の SAS/ACCESS ソフトウェアのドキュメントを参照してください。

動的 LIBNAME Engine

SAS/ACCESS LIBNAME ステートメント

SAS バージョン 7 以降では、使用する DBMS に応じて、SAS ライブラリ参照名を、データベース、スキーマ、サーバー、テーブルや SAS ビューのグループに直接関連付けることができます。DBMS データにライブラリ参照名を割り当てるには、SAS/ACCESS ソフトウェアの LIBNAME ステートメントを使用する必要があります。このステートメントの構文とオプションは、Base SAS ソフトウェアの LIBNAME ステートメントとは異なります。たとえば、Oracle データベースに接続するには、次のような SAS/ACCESS の LIBNAME ステートメントを使用します。

```
libname mydblib oracle user=smith password=secret path='myoracleserver';
```

この LIBNAME ステートメントは、接続オプション USER=、PASSWORD=、PATH=を指定することで、Oracle に接続します。接続オプション以外にも、実行するデータベースの接続タイプを制御するための SAS/ACCESS ソフトウェアの LIBNAME ステートメントの各種オプションを指定できます。また、追加オプションを使用すると、データの処理方法を制御できます。

ライブラリ参照名と割り当てられている DBMS データを表示および更新するには、DATA ステップ、SAS プロシジャ、**エクスプローラ**ウィンドウのいずれかを使用します。DBMS オブジェクトの情報を表示するには、DATASETS プロシジャおよび CONTENTS プロシジャを使用します。

DBMS データを参照するライブラリ参照名と組み合わせて使用できる SAS/ACCESS ソフトウェアの LIBNAME ステートメントの各種オプションの一覧については、SAS/ACCESS ソフトウェアのドキュメントを参照してください。

SAS/ACCESS ライブラリ参照名と組み合わせて使用できるデータセットオプション

DBMS データにライブラリ参照名を割り当てた後、データに対して、SAS/ACCESS のデータセットオプション、および一部の Base SAS のデータセットオプションを使用できます。次の例では、DB2 データにライブラリ参照名を関連付けて、SQL プロシジャを使用してデータのクエリを行います。

```
libname mydb2lib db2;

proc sql;
   select *
      from mydb2lib.employees(drop=salary)
      where dept='Accounting';
quit;
```

LIBNAME ステートメントにより、DB2 に接続します。ライブラリ参照名と DBMS オブジェクト名で構成される 2 レベル名を指定すると、DBMS オブジェクト(この場合は DB2)を参照できます。DROP=データセットオプションによって、クエリが返すデータから、DB2 のテーブル EMPLOYEES の SALARY 列が除外されます。

DBMS データを参照するデータセットに対して使用できる SAS/ACCESS データセットオプションおよび Base SAS データセットオプションの一覧については、SAS/ACCESS ソフトウェアのドキュメントを参照してください。

PROC SQL ビュー内に SAS/ACCESS LIBNAME ステートメントを埋め込む

SAS/ACCESS の LIBNAME ステートメントは、前述の例のように単独で実行するか、PROC SQL で CREATE VIEW ステートメントの一部として実行することができます。CREATE VIEW ステートメントの USING 句を使用すると、SAS/ACCESS LIBNAME ステートメントを SAS ビューに埋め込むことで、DBMS 接続情報を SAS ビューに格納することができます。次の例では、埋め込まれた SAS/ACCESS の LIBNAME ステートメントを使用しています。

```
libname viewlib 'SAS-library';

proc sql;
   create view viewlib.emp_view as
      select *
         from mydblib.employees
         using libname mydblib oracle user=smith password=secret
            path='myoraclepath';
quit;
```

SQL プロシジャによって SAS ビューが実行されると、SELECT ステートメントによって、ライブラリ参照名が割り当てられ、DBMS への接続が確立されます。ライブラリ参照名の範囲は、その SAS ビューに限定されます。このため、同一の SAS セッション中に同じ名前のライブラリ参照名があっても衝突しません。クエリが終了すると、接続が終了し、ライブラリ参照名の割り当てが解除されます。

注: PROC SQL ビューに Base SAS ソフトウェアの LIBNAME ステートメントを埋め込むこともできます。

SQL プロシジャのパススルー機能

SQL プロシジャのパススルー機能は、SQL プロシジャの拡張機能です。この機能を使用すると、DBMS 固有のステートメントを DBMS に送信して、DBMS データを取得することができます。パススルー機能を使用する場合は、SAS SQL 構文の代わりに、DBMS SQL 構文を指定します。パススルー機能のステートメントを、PROC SQL クエリで使用するか、PROC SQL ビューに格納することができます。

パススルー機能は、次の 3 つのステートメントと 1 つの句で構成されます。

- CONNECT ステートメントにより、DBMS との接続を確立します。
- EXECUTE ステートメントにより、動的でクエリでない DBMS 固有の SQL ステートメントを DBMS に送信します。
- SQL プロシジャの SELECT ステートメントの FROM 句における CONNECTION TO 句により、DBMS からデータを直接取得します。
- DISCONNECT ステートメントにより、DBMS との接続を終了します。

次のパススルー機能の例では、ORACLE データベースにクエリを送信して処理します。

```
proc sql;
   connect to oracle as myconn (user=smith password=secret
      path='myoracleserver');

   select *
```

```
      from connection to myconn
         (select empid, lastname, firstname, salary
            from employees
            where salary>75000);

   disconnect from myconn;
quit;
```

この例では、パススルー機能の CONNECT ステートメントを使用し、USER=、PASSWORD=、PATH=オプションの引数の値を指定して、ORACLE データベースとの接続を確立しています。SELECT ステートメントの FROM 句で CONNECTION TO 句を使用すると、データベースからデータを取得できます。Oracle に送信する DBMS 固有のステートメントは、かっこで囲みます。DISCONNECT ステートメントは、Oracle との接続を終了します。

同じクエリを PROC SQL ビューに格納するには、次の CREATE VIEW ステートメントを使用します。

```
libname viewlib
'SAS-library';

proc sql;
   connect to oracle as myconn (user=smith password=secret
      path='myoracleserver');

   create view viewlib.salary as
      select *
         from connection to myconn
            (select empid, lastname, firstname, salary
               from employees
               where salary>75000);

   disconnect from myconn;
quit;
```

ACCESS プロシジャとインターフェイスビューエンジン

ACCESS プロシジャを使用すると、アクセスディスクリプタを作成できます。アクセスディスクリプタは、メンバタイプ ACCESS を持つ SAS ファイルとして管理されます。これは、DBMS に格納されているデータを、SAS System で認識できる形式で記述したものです。アクセスディスクリプタを使用すると、ビューディスクリプタと呼ばれる SAS/ACCESS ビューを作成できます。ビューディスクリプタは、メンバタイプ VIEW を持つファイルであり、PROC SQL で作成される SAS ビューと同じように機能します。詳細については、“PROC SQL ビュー内に SAS/ACCESS LIBNAME ステートメントを埋め込む” (701 ページ) および “SQL プロシジャのパススルー機能” (701 ページ)を参照してください。

注: 使用する DBMS に対して動的 LIBNAME Engine を利用できる場合は、アクセスディスクリプタおよびビューディスクリプタの代わりに、SAS/ACCESS ソフトウェアの LIBNAME ステートメントを使用して、DBMS データにアクセスすることをお勧めします。ただし、バージョン 6 で、使用する DBMS に対してディスクリプタを使用できる場合は、ディスクリプタが SAS ソフトウェア内で動作し続けます。ディスクリプタを使用すると、ロングネーム変数名などのような一部の新しい機能がサポートされません。

次の例では、アクセスディスクリプタとビューディスクリプタを同一の PROC ステップで作成し、テーブル DB2 からデータを取得します。

```
libname adlib 'SAS-library';
libname vlib 'SAS -library';

proc access dbms=db2;
   create adlib.order.access;
   table=sasdemo.orders;
   assign=no;
   list all;

   create vlib.custord.view;
   select ordernum stocknum shipto;
   format ordernum 5.
          stocknum 4.;
run;

proc print data=vlib.custord;
run;
```

アクセスディスクリプタおよびビューディスクリプタを使用する場合、DBMS のデータを取得する前に、両方のタイプのディスクリプタを作成しておく必要があります。最初にアクセスディスクリプタを作成しておくことで、照会する特定の DBMS テーブルに関する情報が格納できるようになります。

アクセスディスクリプタを作成したら、次に、1 つまたは複数のビューディスクリプタを作成し、アクセスディスクリプタに記述された DBMS データの一部またはすべてを取得します。ビューディスクリプタでは、変数を選択し、形式を適用して、SAS System でデータを表示、出力、格納します。SAS プログラムでは、ビューディスクリプタのみを使用し、アクセスディスクリプタは使用しません。

インターフェイスビューエンジンを使用すると、例にある PROC PRINT ステップのように DATA ステップまたは PROC ステップで 2 レベルの SAS 名を使用して、SAS ビューを参照することができます。

SAS ビューの詳細については、27 章, “SAS ビュー” (665 ページ) を参照してください。アクセスディスクリプタと SAS/ACCESS ビューの作成および使用に関する詳細については、各 DBMS 用の SAS/ACCESS ドキュメントを参照してください。

DBLOAD プロシジャ

DBLOAD プロシジャを使用すると、SAS データセット、データファイル、SAS ビュー、DBMS テーブルからデータを作成して別の DBMS テーブルにロードすることや、既存のテーブルに行を追加することができます。また、クエリでない DBMS 固有の SQL ステートメントを、SAS セッションから DBMS にサブミットすることもできます。

注: 使用する DBMS に対して動的 LIBNAME Engine を利用できる場合は、DBLOAD プロシジャの代わりに、SAS/ACCESS ソフトウェアの LIBNAME ステートメントを使用して DBMS データを作成することをお勧めします。ただし、バージョン 6 で、使用する DBMS に対して DBLOAD プロシジャが利用可能であった場合、同プロシジャは SAS ソフトウェア内で動作し続けます。ただし、DBLOAD プロシジャを使用すると、ロングネーム変数名などのような一部の新しい機能がサポートされません。

次の例では、前に作成した SAS データセット INVDATA から、ORACLE データベースのテーブル INVOICE にデータを追加します。:

```
proc dbload dbms=oracle data=invdata append;
  user=smith;
  password=secret;
  path='myoracleserver';
  table=invoice;
  load;
run;
```

DBLOAD プロシジャの詳細については、各 DBMS 用の SAS/ACCESS ドキュメントを参照してください。

インターフェイス DATA ステップエンジン

一部の SAS/ACCESS ソフトウェア製品は、DATA ステップインターフェイスをサポートします。DATA ステップインターフェイスを使用すると、DATA ステッププログラムを使用して DBMS からデータを読み込むことができます。SAS ソフトウェアプロダクトの中には、DATA ステップインターフェイスでの読み込みと書き出しの両方をサポートしているものがあります。

DATA ステップインターフェイスは、次の 4 つのステートメントで構成されます。

- INFILE ステートメントにより、アクセスするデータベースまたはメッセージキューを識別します。
- INPUT ステートメントと INFILE ステートメントにより、GET 呼び出しを発行し DBMS データを取得します。
- FILE ステートメントにより、更新するデータベースまたはメッセージキューを識別します(DBMS への書き出しがサポートされている場合)。
- PUT ステートメントと INFILE ステートメントにより、UPDATE 呼び出しを発行します(DBMS への書き出しがサポートされている場合)。

次の例は、DATA ステップで FILE ステートメントと INFILE ステートメントを使用して、IMS データベースのデータを更新します。これらのステートメントは、IMS 固有言語の DL/I で、データベースへの呼び出しを生成します。次の DATA ステップでは、新しい顧客に関する情報を含む既存の SAS データセット Bank.Customer を読み取った後、その SAS データセット内のデータを使用して ACCOUNT データベースを更新します。

```
data _null_;
  set bank.customer;
  length ssa1 $9;
  infile accupdt dli call=func dbname=db ssa=ssa1;
  file accupdt dli;
  func = 'isrt';
  db = 'account';
  ssa1 = 'customer';
  put @1   ssnumber $char11.
      @12  custname $char40.
      @52  addr1    $char30.
      @82  addr2    $char30.
      @112 custcity $char28.
      @140 custstat $char2.
      @142 custland $char20.
      @162 custzip  $char10.
      @172 h_phone  $char12.
      @184 o_phone  $char12.;
```

```
  if _error_ = 1 then
    abort abend 888;
run;
```

DATA ステップインターフェイスを提供する SAS/ACCESS 製品では、INFILE ステートメントで、特殊な DBMS 固有のオプションを使用できます。このオプションを使用すると、DBMS の変数値を指定すること、および DBMS の呼び出しを適切にフォーマットすることができます。DBMS に対して使用できる、DBMS 固有の INFILE ステートメントオプションおよび Base SAS ソフトウェアの INFILE ステートメントオプションの一覧については、各 DBMS 用の SAS/ACCESS のドキュメントを参照してください。

32 章

クロス環境データアクセス(CEDA)を用いたデータ処理

クロス環境データアクセス(CEDA)の定義

クロス環境データアクセス(CEDA)は、Base SAS の機能です。CEDA は、ディレクトリベースの動作環境(UNIX や Windows など)で作成された SAS ファイルを、互換性のない環境で処理することや、互換性のないセッションエンコーディングに従って処理することを可能にします。CEDA では、処理は自動的に行われるため、ユーザーがその処理を意識することはありません。ユーザーは移送ファイルを作成する必要はありません。ファイルを変換する SAS プロシジャを使用するか、または SAS プログラムを変更します。CEDA は、バージョン 7 以降の SAS System で作成されたファイルをサポートします。このドキュメントでは、CEDA がもたらす利点、CEDA に関連する制限事項、および CEDA 処理の動作について説明します。

CEDA に関する用語を次に示します。

データ表現
: 特定の動作環境におけるデータの保存形式です。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)、ワードアラインメント(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。

エンコーディング
コンピュータによって処理可能な数値(これをコードポイントと呼ぶ)へとマッピングされている文字(文字、表語文字、数字、区切り文字、記号、制御文字などを含む)の集合です。コードポイントは、エンコーディング方式を適用することにより、特定の文字セット内の文字へと割り当てられます。エンコーディングの例としては、Wlatin1 や Danish EBCDIC などが挙げられます。

非互換ファイル
現在の SAS セッションとは異なるデータ表現やエンコーディングを持つファイルのことです。CEDA を使うと、多くの種類の非互換ファイルにアクセスできるようになります。

CEDA の利点

CEDA には次の利点があります。

- ユーザーは、サポートされている SAS ファイルを透過的に処理できます。ファイルのデータ表現や文字エンコーディングに関する知識は必要ありません。
- 移送ファイルは作成されません。CEDA は、現在のセッションのデータ表現の単一ステップによる変換を行います。
- CEDA では、ソースとなるデータ表現から移送ファイルへ、そして移送ファイルからターゲットとなるデータ表現へというような、ファイルを処理するための複数のステップは必要ありません。
- CEDA では、SAS/CONNECT ソフトウェアで必要となるサインオンも、SAS/SHARE で必要となる専用サーバーも必要ありません。

CEDA を使用した SAS ファイルの処理

CEDA がサポートする処理

CEDA は、ディレクトリベースの動作環境(UNIX、Windows、OpenVMS など)で作成されたバージョン 7 以降の SAS ファイルをサポートします。CEDA は、次に示す SAS エンジンで、次の表に示す各種の SAS ファイルの処理を提供します。

BASE
SAS バージョン 9 (V9)、SAS バージョン 8 (V8)、SAS バージョン 7 (V7)で提供されるデフォルト Base SAS Engine です。

SASESOCK
SAS/CONNECT ソフトウェアで提供される TCP/IP ポート用のエンジンです。

TAPE
SAS バージョン 9 (V9)、SAS バージョン 8 (V8)、SAS バージョン 7 (V7)で提供されるシーケンシャルエンジンです。

表 32.1 CEDA が提供する SAS ファイル処理

SAS ファイルのタイプ	エンジン	サポートされている処理
SAS データファイル	BASE、TAPE、SASESOCK	入出力*
PROC SQL ビュー	BASE	入力
Oracle または Sybase 向けの SAS/ACCESS ビュー	BASE	入力
MDDB ファイル**	BASE	入力

* 既存の SAS データファイルを置き換える出力処理の場合、動作に違いがあります。詳細については、"出力処理の動作の違い" (709 ページ)を参照してください。

** CEDA は、SAS バージョン 8 以降の MDDB ファイルをサポートします。

出力処理の動作の違い

For output processing that replaces an existing SAS data file, the BASE and TAPE engines behave differently regarding the following attributes:

encoding

- BASE Engine は、ソースライブラリ内にあるファイルのエンコーディングを使用します。すなわち、エンコーディングがクローン化されます。
- TAPE Engine は、現在の SAS セッションのエンコーディングを使用します。
- BASE Engine と TAPE Engine のどちらを使用する場合でも、COPY プロシジャは、デフォルトでソースライブラリ内にあるファイルのエンコーディングを使用します。COPY プロシジャで現在の SAS セッションのエンコーディングを使用したい場合、NOCLONE オプションを指定する必要があります。別のエンコーディングを使用したい場合、NOCLONE オプションとともに ENCODING=オプションを指定します。SAS/SHARE ソフトウェアまたは SAS/CONNECT ソフトウェアで COPY プロシジャを使用する場合、デフォルトでは、現在の SAS セッションのエンコーディングが使用されます。

データ表現

- BASE Engine および TAPE Engine は、COPY プロシジャの場合を除き、現在の SAS セッションのデータ表現を使用します。
- BASE Engine と TAPE Engine のどちらも使用する場合、COPY プロシジャは、デフォルトでソースライブラリ内にあるファイルのデータ表現を使用します。COPY プロシジャで現在の SAS セッションのデータ表現を使用したい場合、NOCLONE オプションを指定します。別のデータ表現を使用したい場合、NOCLONE オプションと共に OUTREP=オプションを指定します。SAS/SHARE ソフトウェアまたは SAS/CONNECT ソフトウェアで COPY プロシジャを使用する場合、デフォルトでは、現在の SAS セッションのデータ表現が使用されます。

CEDA の制限

CEDA には次の制限事項があります。

- SAS カタログはサポートされていません。カタログエントリには、出力形式、コンパイル済みストアドマクロ、SAS/AF アプリケーション、SAS/GRAPH 出力、SAS コー

ド、SCL コード、データ、各種 SAS プロシジャに固有のその他のエントリタイプを含みます。

- 更新処理はサポートされません。
- 一貫性制約は読み取りや更新はできません。
- 監査証跡ファイルは更新できません。ただし、読み取りは可能です。
- インデックスはサポートされません。したがって、インデックスを使用した WHERE 句の最適化はサポートされません。
- 拡張属性は更新できません。ただし、読み取りは可能です。
- その他のサポートされていないファイルとしては、DATA ステップビュー、SAS/ACCESS ビュー(Oracle または Sybase 向けの SAS/ACCESS の場合を除く)、コンパイル済みストアド DATA ステッププログラム、アイテムストア、DMDB ファイル、FDB ファイル、SAS 7 より前のバージョンで作成された SAS ファイルすべてが挙げられます。
- z/OS 上では、任意の SAS データ表現を使用して UNIX ファイルシステムライブラリのメンバを作成できます。ただし、連結ライブラリを作成する場合、ライブラリを作成する SAS セッションのデータ表現がそれらに割り当てられます。連結ライブラリの場合、そのライブラリのデータ表現と異なるデータ表現(文字エンコーディングは除く)を持つライブラリメンバを作成することはできません。たとえば、z/OS 上の 31 ビット版 SAS で連結ライブラリを作成する場合、同ライブラリは MVS_32 のデータ表現を持ちます。また、LIBNAME ステートメントの OUTREP オプションを使用して、MVS_32 以外のデータ表現を持つメンバを同ライブラリ内に作成することはできません。z/OS 上の BASE Engine およびシーケンシャルエンジンにおけるライブラリの実装については、*z/OS 版 SAS* を参照してください。
- BASE Engine はデータの読み込み時に同データを変換するため、複数のプロシジャを実行すると、SAS System によるデータの読み込みと変換が複数回必要となります。このため、変換の回数が多すぎると、システムパフォーマンスに影響する場合があります。
- データセットが破損した場合、CEDA はそれを修復するためのファイル処理を行えません。これは、破損したデータセットの修復には更新処理が必要となりますが、CEDA は更新処理をサポートしていないためです。ファイルを修復するには、ファイルが作成された環境か、または CEDA 処理を起動しない、互換性のある環境に対象のファイルを移動する必要があります。破損したデータセットを修復する方法の詳細については、*Base SAS Procedures Guide* にある DATASETS プロシジャの REPAIR ステートメントを参照してください。
- 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。エンコーディングとトランスコーディングの詳細については、*SAS National Language Support (NLS): Reference Guide* を参照してください。
- 異なる動作環境間でデータを移動した場合、数値変数の精度が失われることがあります。その場合、数値変数が短い変数長で定義されているならば、変数長を増やすと良いでしょう。フルサイズの数値変数では、CEDA による精度の損失が発生する可能性は低くなります。詳細については、"SAS における数値の正確さ" (58 ページ)を参照してください。
- 数値変数の最小長は、動作環境によって、2 バイトか 3 バイトのどちらかになります。3 バイトの最小長をサポートする動作環境(Windows や UNIX など)では、CEDA は、2 バイトの最小長をサポートする環境(z/OS など)で作成された数値変数を処理できません。この制限を解決するには、CEDA のかわりに、XPORT Engine を使用するか、または CPORT プロシジャおよび CIMPORT プロシジャを使用します。

注: 詳細については、*Base SAS Procedures Guide* を参照してください。旧バージョンの SAS System でファイルを作成しているために前述の制限が問題となる場合、MIGRATE プロシジャの使用を検討することをお勧めします。MIGRATE プロシジャを使うと、一貫性制約、インデックス、監査証跡などの多くの機能を保持できます。

CEDA を使用したファイルの処理が行われる場合

CEDA 変換はユーザーにとって透過的であるため、ユーザーはいつ CEDA が使用されるかを意識することはありません。ただし、いつ CEDA が使用されるかを理解することが役に立つ場合もあります。たとえば、CEDA 変換により追加のリソースが必要となる場合、いつ CEDA が使用されるかを知ることはユーザーにとって有益となります。

デフォルトでは、CEDA が使用された場合、ログにメッセージが書き出されます。次に例を示します。

ログ 32.1 *異なる動作環境からのファイル処理によるログ出力*

```
NOTE:Data file HEALTH.OXYGEN.DATA is in a format that is native to another host,
or the file encoding does not match the session encoding.Cross Environment Data
Access will be used, which might require additional CPU resources and might
reduce performance.
```

CEDA は次の場合に使用されます。

- SAS ファイルで使用されている文字値のエンコーディングが、現在実行中の SAS セッションのエンコーディングと互換性がない場合。
- SAS ファイルのデータ表現が、現在実行中の SAS セッションのデータ表現と互換性がない場合。たとえば、Windows 環境から UNIX 環境へとファイルを移動した場合や、32 ビットの UNIX 環境から 64 ビットの UNIX 環境へとアップグレードした場合に、このような非互換性が発生します。

  次の表の各行には、互換性のある動作環境のグループが示されています。CEDA が使用されるのは、次の表のある行に含まれているデータ表現で作成したファイルを、別の行に含まれているデータ表現で処理する場合に限られます。(SAS の現在のリリースの実行環境ではないものもありますが、完全を期するため、ここにはその環境も含めています。)

表 32.2 *環境間の互換性*

データ表現値	動作環境
ALPHA_TRU64	Tru64 UNIX *
LINUX_IA64	Itanium ベースシステム向け Linux*
LINUX_X86_64	x64 向け Linux *
SOLARIS_X86_64	x64 向け Solaris *
ALPHA_VMS_32	OpenVMS Alpha **
ALPHA_VMS_64	OpenVMS Alpha **
VMS_IA64	HP Integrity 上の OpenVMS**

データ表現値	動作環境
HP_IA64	Itanium プロセッサファミリアーキテクチャ向け HP-UX
HP_UX_64	PA-RISC 向け HP-UX(64 ビット版)
RS_6000_AIX_64	AIX
SOLARIS_64	SPARC 向け Solaris
HP_UX_32	PA-RISC 向け HP-UX
MIPS_ABI	MIPS ABI
RS_6000_AIX_32	AIX
SOLARIS_32	SPARC 向け Solaris
LINUX_32	Intel アーキテクチャ向け Linux
INTEL_ABI	Intel アーキテクチャ向け ABI
MVS_32	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
VAX_VMS	OpenVMS VAX
WINDOWS_32	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)***

* 本グループ内の 4 つの環境にはすべて互換性がありますが、カタログは例外です。カタログは、Tru64 UNIX と Linux for Itanium-based systems 間で互換性があります。カタログは、Tru64 UNIX と Linux for Itanium-based systems 間で互換性があります。

** これらの OpenVMS 環境は一部のコンパイラタイプでは異なるデータ表現を持ちますが、BASE Engine により作成された SAS データセットは、それらの異なるデータ型を格納しません。このため、エンコーディングが互換である場合、これらの環境間で CEDA は使用されません。ただし、バージョン 9 の SAS System は、OpenVMS 環境で作成された SAS バージョン 8 のカタログをサポートしないことに注意してください。カタログを移行するには、MIGRATE プロシジャを使用します。詳細については、*Base SAS Procedures Guide* を参照してください。

*** これらの Windows 環境には互換性がありますが、カタログは例外です。Windows 版 SAS System の 32 ビット版と 62 ビット版の間では、カタログは非互換になります。

更新処理を許可するかどうかの設定

ファイルのデータ表現が処理環境のデータ表現と同じである場合、エンコーディングが現在実行中の SAS セッションのエンコーディングと互換であるならば、そのファイルを変換するために CEDA は必要とされないため、ユーザーは同ファイルを手動で更新できます。たとえば、64 ビットの Solaris 環境でファイルを作成した場合や、OUTREP= オプションを使用してファイルにデータ表現を指定した場合、ユーザーはそのファイルを、Solaris for SPARC、Solaris for HP-UX、Solaris for AIX 上の 64 ビット版 SAS セッションで更新できます。

それ以外の場合、CEDA を使用してファイルの変換が行われるならば、同ファイルは更新できません。ユーザーが同ファイルを更新しようとすると、更新は禁止されているというエラーメッセージが表示されます。次に例を示します。

```
ERROR:File HEALTH.OXYGEN cannot be updated because its encoding does not match
the session encoding or the file is in a format native to another host, such as
HP_UX_64, RS_6000_AIX_64, SOLARIS_64, HP_IA64.
```

ファイルで使われているデータ表現やエンコーディングを判定するには、CONTENTS プロシジャ(または DATASETS プロシジャの CONTENTS ステートメント)を使用します。たとえば、データセット HEALTH.OXYGEN は UNIX 環境の SAS バージョン 9 で作成されたとします。同ファイルを Windows 環境の SAS バージョン 9 へと移動した場合、CONTENTS プロシジャの出力は次のようになります。

アウトプット 32.1 データ表現を示す CONTENTS プロシジャ出力

The CONTENTS Procedure

Data Set Name	HEALTH.OXYGEN	**Observations**	40
Member Type	DATA	**Variables**	7
Engine	V9	**Indexes**	0
Created	04/10/2014 15:54:03	**Observation Length**	56
Last Modified	04/10/2014 15:54:03	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	HP_UX_64, RS_6000_AIX_64, SOLARIS_64, HP_IA64		
Encoding	latin1 Western (ISO)		

CEDA 以外の方法

CEDA が持つ制限が原因で、CEDA が使用できない場合があります。そのような場合、異なる動作環境間でファイルを移動するには、次の方法を使用します。

DATA ステップまたは COPY プロシジャで XPORT Engine を使用
: 移動元の環境で、DATA ステップか COPY プロシジャのいずれかを使用して、XPORT Engine を指定した LIBNAME ステートメントを実行することにより、SAS データセットから移送ファイルを作成します。移動先の環境で前述と同じ方法を使用して、移送ファイルを移動先の環境で使用される出力形式へと変換します。XPORT Engine は、SAS バージョン 7 以降の機能(長いファイル名や長い変数名)をサポートしていないことに注意してください。

DATA ステップまたは COPY プロシジャで XML Engine を使用
: 移動元の環境で、DATA ステップか COPY プロシジャで、XML Engine を指定した LIBNAME ステートメントを実行することにより、SAS データセットから XML ファイルを作成します。移動先の環境で前述と同じ方法を使用して、XML ドキュメントを移動先の環境で使用される出力形式へと変換します。

CPORT プロシジャおよび CIMPORT プロシジャ
移動元の環境で、CPORT プロシジャによりデータセットまたはカタログを移送形式へと書き出します。移動先の環境で PROC CIMPORT を使用して、移送ファイルを移動先の環境で使用される出力形式へと変換します。

SAS/CONNECT ソフトウェアにおけるデータ転送サービス
データ転送サービスとは、データのディスクコピーを転送し、SAS リリース間で必要な変換に加えて、ある環境のデータ表現を別のデータ表現へ変換するバルクデータ転送メカニズムです。SIGNON コマンドを使用して 2 つの SAS セッション間の接続を確立した後、UPLOAD プロシジャまたは DOWNLOAD プロシジャを実行してデータを移動する必要があります。

SAS/CONNECT ソフトウェアおよび SAS/SHARE ソフトウェアでのリモートライブラリサービス
リモートライブラリサービスを使用すると、LIBNAME ステートメントを使ったリモートデータへの透過的なアクセスを行えます。

データ表現が異なるファイルの作成

デフォルトでは、新規ファイルを作成する場合、SAS を実行している CPU のデータ表現が使用されます。このデフォルトをオーバーライドするには、OUTREP=オプションを指定します。

OUTREP=オプションは、SAS データセットオプションとしても LIBNAME ステートメントのオプションとしても指定できます。データセットオプションは個々のファイルに対して適用されます。一方、LIBNAME ステートメントのオプションは、ライブラリ全体に適用されます。このオプションを指定すると、異なるデータ表現を持つファイルを作成する場合にのみ CEDA が使用されます。

たとえば、UNIX 環境で、データ表現 WINDOWS_32 を持つデータセットを作成したとします。このデータセットを Windows 環境に移動し、Windows 版 SAS (32 ビット)でこのデータセットを処理する場合には、CEDA は呼び出しません。

“例 2:異なるデータ表現での新規ファイルの作成” (716 ページ)および “例 3:既存ファイルのデータ表現の変更” (716 ページ)を参照してください。

詳細については、*SAS Statements: Reference* にある LIBNAME ステートメントの OUTREP=オプションの説明か、または *SAS データセットオプション: リファレンス*の OUTREP=データセットオプションの説明を参照してください。

CEDA の使用例

例 1:ファイルの自動処理

この例は、変換ステップなしでも、異なる動作環境で SAS データセットを簡単に処理できることを示しています。

データセットは最初に UNIX 環境で作成され、後から FTP またはオペレーティングシステムコマンドのようなツールで Windows PC に移動されました。

SAS System は CEDA を使用して自動的に当該ファイルの UNIX データ表現を認識し、それを Windows 環境のデータ表現へ変換します。ログ出力には、CEDA を使用してファイルが処理されるというメッセージが表示されます。

```
libname Health 'c:\MyFiles';

proc print data=Health.Oxygen;
run;
```

ログ 32.2 *異なる動作環境からのファイル処理によるログ出力*

```
NOTE:Data file HEALTH.OXYGEN.DATA is in a format that is native to another host,
or the file encoding does not match the session encoding.Cross Environment Data
Access will be used, which might require additional CPU resources and might
reduce performance.
```

例 2:異なるデータ表現での新規ファイルの作成

この例では、z/OS 動作環境で作業している管理者が、AIX UNIX 環境で処理される UNIX ファイルシステムのディレクトリ内に SAS ファイルを作成するものとします。データセットオプションとして OUTREP=RS_6000_AIX_64 を指定することにより、作成される SAS ファイルのデータ表現を、後でファイルを処理する AIX 動作環境のデータ表現と強制的に一致させます。この方法でファイルを作成すると、後で AIX マシンがファイルを処理する際にデータ変換を行う必要がなくなるため、システムのパフォーマンス向上が期待できます。

```
libname MyLib v9 'HFS-file-spec';

data MyLib.a (outrep=RS_6000_AIX_64);
   infile file-specifications;
   input student $ test1 test2 test3 final;
   total = test1+test2+test3+final;
   grade = total/4.0;
run;
```

例 3:既存ファイルのデータ表現の変更

既存のファイルのデータ表現を変更するには、NOCLONE オプションを指定した COPY プロシジャを使用し、LIBNAME ステートメントで OUTREP=オプションを指定します。次の例では、Windows 形式のデータセットを含むソースライブラリを、Solaris 形式のデータセットを含むターゲットライブラリへコピーしています。データセットを(FTP などを使用して)別のプラットフォームに移動する場合、同データセットをバイナリファイルとして移動する必要があります。

```
libname Target 'target-pathname' outrep=solaris_x86_64;
libname Source 'source-pathname';
proc copy in=Source out=Target noclone memtype=data;
run;
```

詳細については、*SAS Statements: Reference* にある LIBNAME ステートメントの OUTREP=オプションの説明か、または *SAS データセットオプション: リファレンス*の OUTREP=データセットオプションの説明を参照してください。

33 章

SAS 9.4 における、以前のリリースの SAS ファイルとの互換性

バージョン間の互換性について

SAS バージョン 9 のユーザーは以前のバージョンで作成した既存のデータやプログラムを所有しています。このため、既存の SAS ファイルをシームレスに処理することや、SAS バージョン 9 とそれ以前のバージョンの両方を同時に運用することが必要となる場合があります。多くの場合、SAS バージョン 9 では、バージョン 8、7、6 で作成した SAS ファイルを、事前にファイル変換せずに使用できます。ただし、いくつかの制限事項があります。

バージョン間で互換性があるかどうかは、SAS ファイルの種類、現在実行している SAS リリース、ファイルが作成された動作環境、実行する処理のタイプにより異なります。互換性の問題は、通常、SAS System により自動的に処理されます。ただし状況によっては、ユーザーによるエンジン名の指定や、ファイルの移行が必要となる場合があります。

本セクションでは、互換性に関する概要を示します。

ファイルの移行に関する具体的な手順やガイドラインについては、**`support.sas.com`** の Migration Focus Area を参照してください。

Migration Focus Area には、SAS の以前のバージョンから SAS バージョン 9 へとファイルを移行するためのガイダンス情報が示されています。プランニングやコスト分析、互換性に関する既知の問題、ステップごとの移行手順については、Migration Focus Area を参照してください。MIGRATE プロシジャを使用すると、以前のバージョンで作成された複数の SAS ファイルを含むライブラリを簡単に移行できます。詳細については、*Base SAS Procedures Guide* を参照してください。

SAS 9 と以前のリリースの比較

SAS 9 のファイル形式

SAS 7 や SAS 8 では長いファイル名や変数名を使用できるようになったため、ファイルの形式が SAS 6 とは異なっています。

SAS バージョン 9 では、ファイルの形式は基本的にバージョン 7 および 8 と同じです。Base SAS Engine も基本的に同じですが、バージョン 9 では長い出力形式名および入力形式名を定義することや、1 つの SAS データセット内に 32,767 個を超える数の変数を含めることができます。これらの機能は SAS バージョン 7 や 8 では使用できません。

SAS バージョン 7 および 8 で作成された SAS ファイルは、SAS バージョン 9 と互換性があります。ただし、32 ビット版の SAS System で作成された SAS ファイルは、64 ビット版の SAS System で作成された SAS ファイルとは異なるデータ表現を持ちます。データ表現とは、コンピュータアーキテクチャ上や動作環境におけるデータの表示形式のことです。32 ビット版の SAS System で作成された SAS ファイルを 64 ビット版の SAS System で処理する場合には、データ表現に関する制限事項があります。

EXTENDOBSCOUNTER=データセットオプションを使用して、SAS 9.4 で生成されたファイルとそれ以前のリリースのファイルとの互換性の問題を解決することもできます。詳細については、次の文書を参照してください。

- “64 ビット動作環境でのオブザベーションカウントの拡張” (661 ページ)
- “EXTENDOBSCOUNTER= Data Set Option” (*SAS Data Set Options: Reference*)

SAS 9 のファイル拡張子

ファイル拡張子は、ファイルおよび SAS ファイルのメンバタイプの両方の作成に使用されたエンジンを反映しています。

SAS System はファイルの種類やバージョンを区別する必要があるため、ファイルの作成時に各ファイルに特有の拡張子を自動的に割り当てます。たとえば、SAS バージョン 7 およびバージョン 8 のファイルは、SAS バージョン 6 のファイルとは拡張子が異なります。

SAS バージョン 9 では、ファイルの拡張子は SAS バージョン 7 および 8 のものと同じです。

各動作環境における SAS データファイル(メンバタイプが DATA である SAS データセット)のファイル拡張子が、SAS System バージョン 6、7、8、9 間でどのように異なっているかを下記の表に示します。

表 33.1 *各動作環境における SAS データファイルの拡張子*

エンジン名	UNIX	OpenVMS for Integrity Server	Windows	z/OS*
V6	.ssd01	.SASEB$DATA	.sd2	利用不可
V7	.sas7bdat	.sas7bdat	.sas7bdat	.sas7bdat

エンジン名	UNIX	OpenVMS for Integrity Server	Windows	z/OS*
V8	.sas7bdat	.sas7bdat	.sas7bdat	.sas7bdat
V9	.sas7bdat	.sas7bdat	.sas7bdat	.sas7bdat

* UNIX System Services の階層ファイルシステム内に存在する SAS データセットに適用されます。

動作環境の情報
SAS メンバタイプや拡張子の詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

SAS ライブラリエンジンの使用

SAS ライブラリにアクセスするには、ライブラリ参照名とライブラリエンジン名が必要です。ライブラリ参照名を SAS ライブラリに割り当てるには、LIBNAME ステートメントか**ライブラリの作成**ウィンドウを使用しますが、ユーザーは通常エンジン名を指定する必要はありません。これは、SAS System が適切なエンジンを自動的に選択するためです。

エンジン名を省略した場合は、ライブラリの内容に基づいて、エンジンが自動的に割り当てられます。たとえば、SAS System はバージョン 6 の SAS ライブラリとバージョン 9 の SAS ライブラリを区別できます。SAS バージョン 9 では、バージョン 7 およびバージョン 8 の SAS ファイルを含んでいる SAS ライブラリは、バージョン 9 と同じ SAS ライブラリとして扱われることに注意してください。SAS ファイルの形式は同ファイルを作成するエンジンにより決定されるため、ファイルの形式はバージョン 7、8、9 で同一となります。

たとえば、バージョン 9 の SAS セッションで、次の LIBNAME ステートメントを発行して、バージョン 8 の SAS ファイルを含む SAS ライブラリにライブラリ参照名を割り当てると、自動的にバージョン 9 のエンジンが割り当てられます。

```
libname mylib 'v8-SAS-library';
```

また、バージョン 9 の SAS セッションで、次の LIBNAME ステートメントを発行して、バージョン 6 の SAS ファイルのみを含む SAS ライブラリにライブラリ参照名を割り当てると、自動的に SAS 6 の互換性エンジンが使用されます。

```
libname mylib 'v6-SAS-library';
```

エンジンの割り当ては、次の表に示すとおり、SAS ライブラリの内容に基づいて自動的に行われます。

表 33.2 *SAS 9 でのデフォルトライブラリエンジンの割り当て*

エンジンの割り当て	SAS ライブラリの内容
V9	SAS ファイルなし(ライブラリが空)
V9	バージョン 9 の SAS ファイルのみ
V9	バージョン 8 の SAS ファイルのみ

エンジンの割り当て	SAS ライブラリの内容
V9	バージョン 7 の SAS ファイルのみ
V6	バージョン 6 の SAS ファイルのみ
V9	バージョン 9 の SAS ファイルと、それ以前のバージョンの SAS ファイルの両方を含む場合

注: SAS System はライブラリの内容に基づいて自動的にエンジンを割り当てますが、ユーザーがエンジンを明示的に指定する方がより効率的です。たとえば、LIBNAME ステートメントで次のようにエンジン名を指定すると、使用するエンジンの判定処理が不要になります。

```
libname mylib v6 'v6-SAS-library';
```

SAS エンジンの詳細については、35 章, “SAS エンジン” (737 ページ)を参照してください。

34 章

ファイルの保護

パスワードの定義

SAS System では、SAS ライブラリのメンバにパスワードを割り当てることによって、メンバへのアクセスを制限し、SAS ファイルを保護することができます。パスワードは、CATALOG を除くすべてのメンバタイプの SAS ファイルに割り当てることができます。設定できる保護レベルには、読み取り保護、書き込み保護、変更保護の 3 つがあります。割り当てたパスワードは、SAS ログに 1 文字以上の大文字 X で表示されます。

注: SAS データセットには、物理的な格納方法により SAS データファイルと SAS ビューの 2 種類があります。パスワードの働きは、SAS ビュー(メンバタイプが VIEW)であるか、SAS データファイル(メンバタイプが DATA)であるかによって異なります。両者を特に区別する必要がない場合には、“SAS データセット”という用語を使用しています。

読み取り
SAS ファイルの読み取りを禁止します。

書き込み
SAS ファイル内にあるデータの書き換えを禁止します。SAS データファイルに書き込み保護を設定すると、オブザベーションの追加、変更、削除が禁止されます。

変更
SAS ファイルの削除と置き換えを禁止します。SAS データファイルに変更保護を設定すると、変数の属性の変更、インデックスの作成と削除が禁止されます。

変更保護を設定しても、読み取りアクセスや書き込みアクセスは防止されません。また、書き込み保護を設定しても、読み取りアクセスは防止されません。たとえば、変更パスワードや書き込みパスワードを入力しなくても、変更保護や書き込み保護が設定された SAS データファイルを読み込めます。逆に、読み取り保護や書き込み保護を設定しても、変更アクセスは防止されません。たとえば、読み取り保護や書き込み保護を設定してあっても、読み取り保護のみ、または書き込み保護のみが設定された SAS データセットファイルを削除することは可能です。

適切な権限を持たないユーザーによる読み取り、書き込み、変更操作から SAS ファイルを保護するには、それぞれ読み取り保護、書き込み保護、変更保護の保護レベルを SAS ファイルに割り当てます。パスワードなしで読み取りのみ許可し、データファイルの変更や削除を許可しない場合は、SAS ファイルに書き込み保護と変更保護を割り当てます。SAS ファイルを 1 つのパスワードで完全に保護するには、PW=データセットオプションを使用します。詳細については、“PW=データセットオプションを使用した完全な保護の割り当て” (726 ページ)を参照してください。

注: 読み取り保護のみが設定された SAS データセットを更新する場合、読み取りパスワードを入力する必要があります。これは、SAS System がファイルを開くときの仕様です。

注: メンバタイプが VIEW の SAS ビューの場合は、保護のレベルが多少異なります。“ビューでのパスワードの使用” (727 ページ)を参照してください。

パスワードの割り当て

構文

SAS データセットにパスワードを設定する場合、次のいずれかを使用します。

- DATA ステートメント
- DATASETS プロシジャの MODIFY ステートメント
- 一部のプロシジャの OUT=オプション
- SQL プロシジャの CREATE VIEW ステートメント
- ツールボックス

次に、保護対象とするデータセットに 1 つまたは複数のパスワードを割り当てます。既存のデータセットでも、新規に作成するデータセットでもかまいません。構文例を次に示します。

(password-type=password ... password-type=password>)

ここで、password には、割り当てるパスワードとして 8 文字までの有効な SAS 名を指定します。passwordtype には、割り当てる保護レベルとして次の SAS データセットオプションのいずれかを指定します。

- ALTER=
- PW=
- READ=
- WRITE=

注意:
割り当てるパスワードは、すべて記録しておくことをお勧めします。 パスワードを忘れた場合、SAS System からパスワードを取得することはできません。

DATA ステップを使用したパスワードの割り当て

新しい SAS データファイルを作成する場合、DATA ステップでデータセットオプションを使用することによって、保護対象になっていないメンバにパスワードを割り当てることができます。

次の例では、パスワードなしではデータセットの削除や変更が行えないようにしています。

```
   /* assign a write and an alter password to MYLIB.STUDENTS */
data mylib.students(write=yellow alter=red);
   input name $ sex $ age;
   datalines;
Amy f 25
… more data lines …
;
```

次の例では、パスワードなしではストアドプログラムの読み取りや削除、および対応するソースプログラムの変更が行えないようにしています。

```
   /* assign a read and an alter password to the SAS view ROSTER */
data mylib.roster(read=green alter=red) / view=mylib.roster;
   set mylib.students;
run;

libname stored 'SAS-library-2';

   /* assign a read and alter password to the program file SOURCE */
data mylib.schedule / pgm=stored.source(read=green alter=red);
   … DATA step statements …
run;
```

注: 変更保護されている SAS データセットを新しいデータセットで置き換えた場合、新しいデータセットは古いデータセットに設定されていた変更保護パスワードを継承します。新しいデータセット用に変更保護パスワードを変更するには、DATASETS プロシジャの MODIFY ステートメントを使用します。

既存のデータセットへのパスワードの割り当て

SAS データファイルがすでに存在する場合、DATASETS プロシジャで MODIFY ステートメントを使用することによって、保護対象になっていないメンバにパスワードを割り当てることができます。

```
   /* assign an alter password to STUDENTS */
proc datasets library=mylib;
   modify students(alter=red);
run;
```

プロシジャを用いたパスワードの割り当て

一部のプロシジャでは、OUT=オプションで出力データセットを指定した後にパスワードを割り当てることができます。

```
   /* assign a write and an alter password to SCORE */
proc sort data=mylib.math
   out=mylib.score(write=yellow alter=red);
   by number;
run;
```

SQL プロシジャでは、CREATE TABLE ステートメントまたは CREATE VIEW ステートメントを使用してパスワードを割り当てることができます。

```
   /* assign an alter password to the SAS view BDAY */
proc sql;
   create view mylib.bday(alter=red) as
     query-expression;
```

SAS ウィンドウ環境を用いたパスワードの割り当て

SAS ウィンドウ環境では、パスワードウィンドウを使用することにより、データファイルのパスワードの作成や変更が行えます。ツールボックスからパスワードウィンドウを表示するには、後ろにファイル名を付加して、グローバルコマンドの SETPASSWORD を発行します。これにより、指定したデータファイルのパスワードウィンドウが開きます。

非 SAS 環境でのパスワードの割り当て

SAS パスワードでは、SAS System 以外での SAS ファイルへのアクセスは制御されません。SAS System 以外の環境に置かれている SAS ファイルへのアクセスを制御するには、オペレーティングシステムが提供するユーティリティか、またはファイルシステムのセキュリティ制御機能を使用する必要があります。

パスワードの削除と変更

パスワードを削除または変更するには、DATASETS プロシジャの MODIFY ステートメントを使用します。詳細については、“DATASETS” (*Base SAS Procedures Guide*)を参照してください。

DATA ステップと PROC ステップでのパスワード保護された SAS ファイルの使用

パスワード保護されたファイルにアクセスするには、保護を割り当てたときと同じデータセットオプションを使用します。

-
```
/* Assign a read and alter password to the stored program file*/
/*STORED.SOURCE */
data mylib.schedule / pgm=stored.source
   (read=green alter=red);
   <… more DATA step statements …>
run;

   /*Access password-protected file*/
proc sort data=mylib.score(write=yellow alter=red);
   by number;
run;
```

-
```
/* Print read-protected data set MYLIB.AUTOS */
proc print data=mylib.autos(read=green);
run;
```

-
```
/* Append ANIMALS to the write-protected data set ZOO */
proc append base=mylib.zoo(write=yellow) data=mylib.animals;
run;
```

-
```
/* Delete alter-protected data set MYLIB.BOTANY */
proc datasets library=mylib;
   delete botany(alter=red);
run;
```

パスワードは、保護レベルに関して階層構造になっています。たとえば、ALTER パスワードを指定すると、読み取りアクセスと書き込みアクセスが可能になります。次の例では、3 つの異なるパスワードを使用してデータセット States を作成します。続いて、そのデータセットを読み込んでプロットを作成します。

```
data mylib.states(read=green write=yellow alter=red);
   input density crime name $;
   datalines;
151.4 6451.3 Colorado
… more data lines …
;

proc plot data=mylib.states(alter=red);
   plot crime*density;
run;
```

間違ったパスワードの処理

SAS ウィンドウ環境では、ユーザーがパスワードで保護されているメンバにアクセスする際に指定したパスワードが間違っていると、ダイアログボックスが表示され、適切なパスワードの入力を求められます。このダイアログボックスに入力するテキストは、表示されません。PWREQ=データセットオプションを使用すると、パスワードが入力されなかった場合や間違ったパスワードが入力された場合に、ダイアログボックスを表示するかどうかを制御できます。PWREQ=データセットオプションは、SCL アプリケーションで最も役に立ちます。

バッチモードまたは非対話型ラインモードでは、パスワードで保護されているメンバにアクセスしたとき、指定したパスワードが間違っていると SAS ログにエラーメッセージが表示されます。

対話型ラインモードの場合も、指定したパスワードが間違っていると、パスワードの入力を求められます。パスワードを入力して Enter キーを押すと、処理が続行されます。入力したパスワードが間違っている場合は、SAS ログにエラーメッセージが表示されます。

PW=データセットオプションを使用した完全な保護の割り当て

PW=データセットオプションを指定すると、どの保護レベルにも同じパスワードが割り当てられます。このデータセットオプションは、1 つのパスワードだけでメンバを完全に保護する場合に便利です。PW=データセットオプションを使用すれば、アクセスするユーザーは、どのようなアクセスを行う場合でもパスワードを 1 つ覚えておくだけで済みます。

- PW=データセットオプションによってパスワードが割り当てられているメンバにアクセスするには、PW=データセットオプションを使用します。また、目的のアクセスレベルに相当するデータセットオプションを使用することもできます。

```
   /* create a data set using PW=, then use READ= to print the data set */
data mylib.states(pw=orange);
   input density crime name $;
   datalines;
151.4 6451.3 Colorado
… more data lines …
;

proc print data=mylib.states(read=orange);
run;
```

- PW=データセットオプションは、他のパスワードオプションの別名としても利用できます。次に例を示します。

```
   /* Use PW= as an alias for ALTER=. */
data mylib.college(alter=red);
   input name $ 1-10 location $ 12-25;
   datalines;
Vanderbilt Nashville
Rice       Houston
```

```
Duke        Durham
Tulane      New Orleans
… more data lines …
;

proc datasets library=mylib;
   delete college(pw=red);
run;
```

エンコードされたパスワード

パスワードをエンコードすることにより、パスワードを平文(プレーンテキスト)で指定する必要がない SAS プログラムを作成できます。PWENCODE プロシジャは、エンコーディングを使用してパスワードを隠蔽します。エンコーディングを使用すると、テーブルルックアップ形式を通じて、1 つの文字セットを別の文字セットへと変換できます。エンコードされたパスワードを使うことで、パスワードをそのままの形で表示しないようにできます。ただし、専門知識のある攻撃者であれば、エンコードされたパスワードをデコードできるため、エンコードされたパスワードを使うことで、データセキュリティのニーズがすべて解決されるわけではありません。

エンコードされたパスワードを使用する場合、構文パーサーが同パスワードをデコードすることにより、ファイルへのアクセスが可能となります。エンコードされたパスワードが、プレーンテキストで SAS ログに書き出されることはありません。SAS System が受け付けるパスワード長は 8 文字までです。エンコードされたパスワードをデコードした結果、同パスワードが 8 文字よりも長いことが判明した場合、SAS System はそのパスワードを不正なものとして読み取るため、SAS ログにエラーメッセージが出力されます。詳細については、“PWENCODE” (*Base SAS Procedures Guide*)を参照してください。

ビューでのパスワードの使用

保護レベル

SAS ビューおよびストアドプログラムの保護レベルは、それ以外の種類の SAS ファイル向けの保護レベルと同様です。ただし、SAS ビューでは、パスワードはビューソースデータだけでなく、ビュー定義、ソースステートメントにも適用されます。

SAS ビューに設定できる保護レベルには、読み取り保護、書き込み保護、変更保護の 3 つがあります。次のセクションでは、データセットオプションがビューソースデータやビューのディスクリプタ情報に与える影響について説明します。ここでは、特に断りのない限り、ビューという用語はあらゆるタイプの SAS ビューを指しています。また、ビューソースデータという用語は、SAS ビューによってアクセスされるデータを指しています。

読み取り

- ビューソースデータの読み取りを禁止します。
- DESCRIBE ステートメントを用いた場合に、SAS ログにソースステートメントが表示されないようにします。
- SAS ビューの置き換えを許可します。

書き込み

- SAS ビューに関連付けられたビューソースデータの書き換えを禁止します。

- DESCRIBE ステートメントを用いた場合に、SAS ログにソースステートメントが表示されないようにします。
- SAS ビューの置き換えを許可します。

変更

- DESCRIBE ステートメントを用いた場合に、SAS ログにソースステートメントが表示されないようにします。
- SAS ビューの置き換えを禁止します。

他の SAS ファイルのパスワードと同様に、ビューの読み取り、書き込み、変更パスワードは階層構造になっています。最も制限が強いパスワードが変更パスワードで、最も制限が弱いパスワードが読み取りパスワードです。パスワードで保護されたビューを表示するには、パスワードを指定する必要があります。複数のパスワードを使用してビューが作成されている場合、ビューを表示するには、その最も制限が強いパスワードを使用する必要があります。

たとえば、読み取り保護および書き込み保護されたビューを表示するには、書き込みパスワードを指定する必要があります。同じように、読み取り保護および変更保護されたビューを表示するには、2 つのうち制限が強い方のパスワードである変更パスワードを指定する必要があります。

次のプログラムでは、DESCRIBE ステートメントを使用して、読み取り保護および変更保護されたビューのディスクリプタ情報を表示する方法を示します。

```
   /*create a view with read and alter protection*/
data exam / view=exam(read=read alter=alter);
   set grades;
run;
   /*describe the view by specifying the most restrictive password */
data view=exam(alter=alter);
   describe;
run;
```

ログ 34.1 *パスワード保護されたビュー*

```
NOTE:DATA step view WORK.EXAM is defined as:data exam / view=exam(read=XXX
alter=XXXXX); set grades; run; NOTE:DATA statement used (Total process time):
real time 0.01 seconds cpu time 0.01 seconds
```

詳細については、“DESCRIBE Statement” (*SAS Statements: Reference*)および “DATA Statement” (*SAS Statements: Reference*)を参照してください。

ほとんどの DATA ステップおよび PROC ステップでは、パスワードで保護されているビューを使用する方法は、それ以外のメンバタイプのパスワード保護された SAS ファイルを使用する方法と同じです。たとえば、読み取り保護付きのビューを出力するには次の PRINT プロシジャを実行します。

```
proc print data=mylib.grade(read=green);
run;
```

注: ビューソースデータに対して何らかの保護がすでに設定されている場合、対応する SAS ビューに保護を設定すると、予期しない結果が生じることがあります。

PROC SQL ビュー

通常、パスワードで保護されている SAS データセットを基にして PROC SQL ビューを作成する場合、CREATE VIEW ステートメントの FROM 句の中で、データセットオプションを使用してパスワードを指定します。このようにすると、後でビューを使用するとき

に、パスワードを再指定しなくてもビューソースデータにアクセスできます。たとえば、次のプログラムでは、読み取り保護付きの SAS データセットを基にして PROC SQL ビューを作成しています。ここでは、機密データを保持する変数がビューに含まれないようにしています。

```
proc sql;
   create view mylib.emp as
      select * from mylib.employee(pw=orange drop=salary);
quit;
```

注: パスワードで保護されている SAS データセットの PROC SQL ビューはパスワードを指定せずに作成できます。そのビューを使用するとき、FROM 句に指定されている SAS データセットのパスワードを入力するようにプロンプトが表示されます。バッチモードまたは非対話型モードで SAS System を実行している場合は、エラーメッセージが表示されます。

SAS/ACCESS ビュー

SAS/ACCESS ソフトウェアを利用すると、ビューディスクリプタを編集できます。さらに、一部のインターフェイスではビューソースデータを編集することもできます。ビューディスクリプタの編集と読み込み(表示)を禁止するには、ビューに変更保護を割り当てます。ビューソースデータの更新を禁止するには、ビューに書き込み保護を割り当てます。詳細については、各 DBMS 用の SAS/ACCESS ソフトウェアのドキュメントを参照してください。

DATA ステップビュー

パスワードで保護されている SAS データセットを使用して DATA ステップビューを作成する場合、ビュー定義でパスワードを指定します。このようにすると、ビューを使用するときに、パスワードを再指定しなくてもビューソースデータにアクセスできます。

次のプログラムは、パスワードで保護されている SAS データセットを使用して DATA ステップビューを作成しています。ここでは、機密データを保持する変数がビューに含まれないようにしています。

```
data mylib.emp / view=mylib.emp;
   set mylib.employee(pw=orange drop=salary);
run;
```

このようにすると、SAS ビューはパスワードがなくても使用できますが、ビューソースデータへのアクセスにはパスワードが必要になります。これは、特定のデータ列を保護する方法の一例です。前述の例では、パスワードなしでは、**`proc print data=mylib.emp;`**は実行されますが、**`proc print data=mylib.employee;`**は失敗します。

SAS データファイルの暗号化

SAS データファイルの暗号化について

SAS パスワードとメタデータ連結データセットを使用することで、SAS 内部での SAS データセットへのアクセスを制限できます。しかし、動作環境での SAS データファイルの表示や、外部プログラムによる SAS データファイルの読み込みは阻止できません。暗号化を利用すると、SAS System の範囲を超えて SAS データをセキュリティで保護できます。これは、SAS データを暗号化してディスクに書き込むことで実現します。暗号化

されたデータは、SAS Syatem が同データをディスクから読み込む場合に復号化されます。オペレーティングシステムレベルでの読み込み時や、外部プログラムから読み込まれる場合には、同データは復号化されません。

暗号化は、ファイルへのアクセスには影響しません。ただし、SAS System は、ファイルアクセスを制御するあらゆるホストセキュリティ機構を尊重するため、データセットをメタデータに連結することにより、ホストセキュリティ機構を拡張します。つまり、暗号化と動作環境のセキュリティ機能を併用することができます。

SAS では、データファイルの暗号化には、2 種類のアルゴリズムが使用されます。

- SAS 独自の暗号化 (730 ページ) は、ENCRYPT=YES データセットオプションで実装されます。
- AES (高度暗号化標準)暗号化 (732 ページ) は、ENCRYPT=AES データセットオプションで実装されます。

SAS 9.4 の最初のメンテナンスリリース以降、メタデータ連結ライブラリ管理者は、連結ライブラリ内のすべてのデータファイルがこれら 2 つのアルゴリズムのいずれかを使用して暗号化されていることを要求できるようになりました。詳細については、“Requiring Encryption for Metadata-Bound Data Sets” (*Base SAS Procedures Guide*)および *SAS Guide to Metadata-Bound Libraries* を参照してください。

表 34.1 *暗号化機能*

機能	ENCRYPT=YES	ENCRYPT=AES
ライセンスの必要性	なし	なし
暗号化レベル	中	高
サポートされるアルゴリズム	SAS 独自(Base SAS に同梱)	AES
インストールの必要性	なし(Base SAS の一部)	なし、SAS/SECURE (Base SAS に同梱)
サポートされる動作環境	UNIX Windows z/OS	UNIX Windows z/OS
SAS バージョンサポート	8 以降	9.4 以降

関連項目:

“AUTHLIB” (*Base SAS Procedures Guide*)

SAS 独自の暗号化

SAS 独自の暗号化は Base SAS と一緒にライセンスされる機能であり、すべての配備において使用できます。SAS 独自の暗号化には次の 2 種類があります。

- 32 ビットのローリングキー暗号化手法。これは、パスワードを用いた SAS データセットの暗号化に使用されます。

  この暗号化手法では、SAS データセット内に保存されているパスワードの一部を、当該データの 32 ビットのローリングキーエンコーディングの一部として使用しま

す。この暗号化は、中程度のセキュリティを提供します。ユーザーは、データアクセス認証を受けるために適切なパスワードを提供する必要があります。ただし、この暗号化手法では、2,563,160,682,591 通りの有効なパスワードの組み合わせを試みるような総当たり攻撃(ブルートフォースアタック)を防ぐことはできません。これらの多くは同じ 32 ビットキーを生成する必要があります。SAS/SECURE とデータセットでの AES のサポート(Base SAS に同梱されている機能)を使用することで、より高度なセキュリティを実現できます。

- 32 ビットの固定キー暗号化ルーチン。これは、ログインオブジェクト用のパスワード、構成ファイル内のパスワード、ログインパスワード、内部アカウントパスワードなどの通信に使用されます。

SAS 独自の暗号化を実施するには、ENCRYPT=データセットオプションを使用します。ENCRYPT=データセットオプションを使用できるのは、SAS データファイルの作成時のみです。SAS 独自の暗号化を使用してデータファイルを暗号化する場合、パスワードも割り当てる必要があります。ENCRYPT=YES を指定すると同時に、最低でも、READ=または PW=データセットオプションを指定する必要があります。この暗号化手法ではパスワードが使用されるため、暗号化されたデータセットのパスワードを変更する場合は、そのデータセットの再作成が必要です。

データファイルの暗号化には次の規則が適用されます。

- 暗号化 SAS データファイルをコピーするには、出力エンジンで暗号化がサポートされている必要があります。サポートされていない場合、データファイルはコピーされません。
- 暗号化ファイルは、SAS 6.11 以降のリリースでのみ機能します。
- SAS データビューにはデータが含まれていないため、暗号化はできません。
- データファイルが暗号化される場合、関連付けられたインデックスもすべて暗号化されます。
- 暗号化には、圧縮とほぼ同量の CPU リソースが必要です。
- 暗号化 SAS データファイルに対して PROC CPORT を使用することはできません。

次の例では、SAS 独自の暗号化を使用して SAS データセットを作成します。

```
data salary(encrypt=yes read=green);
   input name $ yrsal bonuspct;
   datalines;
Muriel    34567  3.2
Bjorn     74644  2.5
Freda     38755  4.1
Benny     29855  3.5
Agnetha   70998  4.1
;
```

このデータセットを出力するには、次のように読み取りパスワードを指定します。

```
proc print data=salary(read=green);
run;
quit;
```

関連項目:

“AUTHLIB” (*Base SAS Procedures Guide*)

AES 暗号化

SAS 9.4 では、データセットの AES 暗号化を利用できるようになりました。AES では、最大長 64 文字のキー値を使用することにより、暗号化が強化されています。データセットの作成時に、ENCRYPT=AES を指定します。データセットに格納されているパスワードを使用する SAS 独自の暗号化に対して、AES ではデータセットに格納されていないキー値が使用されます。キー値は、データセットの作成時に、ENCRYPTKEY=データセットオプションを使用して作成されます。AES 暗号化データセットの ENCRYPTKEY=キー値は、データセットを再作成しなければ変更できません。

データセットの AES 暗号化には次の規則が適用されます。

- SAS/SECURE ソフトウェアを使用します。これは Base SAS と一緒にライセンスされる機能であり、すべての配備において使用できます。
- AES 暗号化データセットの作成時またはアクセス時には、ENCRYPTKEY=データセットオプションを使用する必要があります。ただし、メタデータ連結ライブラリ管理者が、データセットに連結されているメタデータ内に暗号化キーを安全に記録している場合は除きます。詳細については、“AUTHLIB” (*Base SAS Procedures Guide*) および *SAS Guide to Metadata-Bound Libraries* を参照してください。
- 暗号化 AES データファイルをコピーするには、出力エンジンで AES 暗号化がサポートされている必要があります。サポートされていない場合、データファイルはコピーされません。
- SAS 9.4 より前のリリースでは、暗号化 AES データファイルは使用できません。
- SAS ビューにはデータが含まれていないため、暗号化はできません。
- 2 つ以上のデータファイルが参照用に関連付けられており、それらのいずれかが AES 暗号化されている場合、それらのファイルをすべて AES 暗号化する必要があります。暗号化キーは、これらのファイルのすべてで同一となります。ただし、メタデータ内に安全に記録されているキーを持つメタデータに連結されているファイルの場合は除きます。メタデータ連結ライブラリの詳細についは、“Metadata-Bound Library” (*Base SAS Procedures Guide*)を参照してください。
- データファイルの AES 暗号化を行う場合、関連付けられたすべてのインデックスも AES 暗号化が行われます。
- AES 暗号化データファイルに対して PROC CPORT を使用することはできません。

ENCRYPTKEY=データセットオプションでは、AES 暗号化ファイルの削除または置換に対する保護はありません。AES 暗号化データセットは、次のシナリオのいずれかを使用すると、暗号化キー値を指定しなくても削除できます。

- PROC DATASETS の KILL オプション
- PROC SQL の DROP ステートメント

暗号化キーでは、ファイルの内容へのアクセスのみが阻止されます。ファイルが削除または置換されないように保護するには、ファイルに ALTER=パスワードを含めるか、またはファイルをメタデータに連結する必要があります。

次の例では、AES 暗号化を使用して暗号化データセットを作成します。

```
data salary(encrypt=aes encryptkey=green);
   input name $ yrsal bonuspct;
   datalines;
Muriel    34567  3.2
Bjorn     74644  2.5
Freda     38755  4.1
Benny     29855  3.5
```

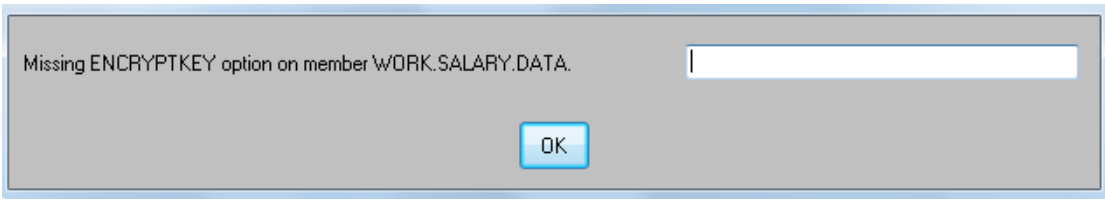
```
Agnetha   70998  4.1
;
```

このデータセットを出力するには、ENCRYPTKEY=キー値を指定します。

```
proc print data=salary(encryptkey=green);
run;
quit;
```

AES 保護データセットのアクセス時に ENCRYPTKEY=キー値を省略すると、ダイアログボックスが表示され、ENCRYPTKEY=キー値の追加を求められます。データセットがメタデータに連結されており、ENCRYPTKEY=キー値が当該ライブラリのメタデータに保存されている場合、ダイアログボックスは表示されません。

図 34.1 *ENCRYPTKEY=のダイアログボックス*



関連項目:
"AUTHLIB" (*Base SAS Procedures Guide*)

AES 暗号化と参照一貫性制約

参照一貫性制約付きデータファイルには、AES 暗号化を使用できます。すべての主キーおよび外部キーデータファイルでは、参照しているすべての外部キーデータファイルおよび主キーデータファイルを開く同一の暗号化キーを使用する必要があります。

世代データセット、監査証跡、インデックス、コピーに対するパスワードと暗号化

SAS System では、パスワードによる保護、SAS 独自の暗号化および AES 暗号化は、保護対象になっているファイルに関連するその他のファイルにも適用されます。そのようなファイルとしては、世代データセット、インデックス、監査証跡、コピーが挙げられます。元のファイルに対応する、保護または暗号化された世代データセット、インデックス、監査証跡、コピーにアクセスできます。元のパスワード保護ファイルまたは暗号化されたファイルを呼び出す場合と同じ規則、構文、動作が適用されます。SAS ビューは、世代データセット、インデックス、監査証跡のいずれも持つことはできません。暗号化の詳細については、"SAS 独自の暗号化" (730 ページ) および"AES 暗号化" (732 ページ)を参照してください。

パスワード値と暗号化キー値の消去

SAS ログを確認

SAS ログを確認して、パスワード値または暗号化キー値がすべて消去されていることを確認する必要があります。これは、READ=、WRITE=、ALTER=、PW=、ENCRYPTKEY=の各オプションに適用されます。

ほとんどの場合、次のように、password=*value* のペアを別々の行に記述して、値を消去します。

```
data &ds(
read=secret
encrypt=aes
encryptkey=evenmoreso
);
x=1;
run;
```

パスワードや暗号化キーが消去されない例

次の例では、SAS ログ内で、パスワード値や暗号化キー値が削除されません。

- DATA ステートメント内のライブラリ参照名やデータセットには、マクロ変数を使用しないでください。

```
%let ds=dataset;

data &ds(read=secret);
   x=1
;
run;
```

次の出力が SAS ログに書き込まれます。

```
111  %let ds=dataset; 112  data &ds(alter=secret); 113     x=1; 114  run; NOTE:The data
```

- 特定のプロシジャ内で不正なパスワードを使用すると、パスワードがログに出力されます。

```
proc append base=here(PW=XXXXXXX) data=more(READ=secret2);
run;
```

- 誤ってタイピングすると、パスワードが SAS ログに出力されます。

```
proc print data=lubrary.abc(READ=secret);
run;
```

または

```
proc print data=library.abc(ERAD=secret);
run;
```

- プログラムによりエラーメッセージが生成された場合、パスワードは消去されません。たとえば、次のコードでは、ライブラリ参照名のスペルを誤って記述しているため、エラーメッセージ"ERROR: libref MYLUB is not assigned."が発行され、パスワードは消去されません。

```
libname mylib 'c:\';
   data mylub.abc(
   read=secret
   );
   x=1;
   run;
```

次の出力が SAS ログに書き込まれます。

```
636  libname mylib 'c:\'; NOTE:Libref MYLIB was successfully assigned as follows:Engine:
```

マクロの使用

マクロ内でパスワードを割り当てた場合、そのパスワードは、同マクロの実行時に SAS ログ内で消去されません。パスワードが SAS ログ内に表示されるのを防ぐには、SAS ログをファイルにリダイレクトします。詳細については、“PRINTTO” (*Base SAS Procedures Guide*)を参照してください。

パスワードの長さ

一部のケースでは、パスワード長に対応する隠し文字の数が 8 文字に固定される場合があります。それ以外の場合、隠し文字の数は、パスワードの長さと同じになります。OPTIONS プロシジャ、VERBOSE オプション、OPLIST オプションの出力では、パスワード長は 8 文字固定になります。

パスワード値が報告される場合、その長さは 8 文字になります。ただし、入力ステートメントに指定されているパスワード値を単にエコーする場合には、元の入力長が保持されます。次の例では、パスワード長を表示します。

```
options pdfpassword=(open=a owner=b );
proc options option=pdfpassword;
run;
```

次の出力が SAS ログに書き込まれます。

```
   634  options pdfpassword=XXXXXXX XXXXXXX X; 635  proc options option=pdfpassword;run; SA
```

メタデータ連結ライブラリ

メタデータ連結ライブラリは、対応するメタデータ保護テーブルオブジェクトに結び付けられる物理的なライブラリです。メタデータ連結ライブラリ内のそれぞれの物理テーブルはそのヘッダーに情報を持ち、ヘッダーが特定のメタデータオブジェクトをポイントしています。そのポインタは、物理的テーブルとメタデータオブジェクト間のセキュリティバインドを形成しています。バインドによって、ユーザーが SAS からのアクセスをどのように要求するかにかかわらず、SAS は物理テーブルのメタデータ層アクセス要件を例外なく確実に適用できます。詳細については、*SAS Guide to Metadata-Bound Libraries* を参照してください。

AUTHLIB プロシジャは、メタデータ連結ライブラリの作成、アクセス、変更に使用されます。このプロシジャの使用は SAS 管理者を対象としています。メタデータ層やホスト層に十分な権限がないユーザーはこのプロシジャを使用できません。詳細については、“AUTHLIB” (*Base SAS Procedures Guide*)を参照してください。

35 章

SAS エンジン

SAS エンジンの定義

エンジンとは、ファイルからの読み取りやファイルへの書き出しを行う SAS ソフトウェアのコンポーネントのことです。各エンジンを使用することで、SAS System は特定の形式のファイルにアクセスできるようになります。エンジンには複数の種類があります。

エンジンの指定

通常、SAS System ではユーザーがエンジンを明示的に指定する必要はありません。エンジン名を省略した場合は、ライブラリの内容に基づいて、エンジンが自動的に割り当てられます。

SAS System はライブラリの内容に基づいて自動的にエンジンを割り当てますが、ユーザーがエンジンを明示的に指定する方がより効率的です。一部の動作環境では、ライブラリの中身を判断するために SAS System が追加の処理手順を実行する必要があります。この場合、SAS System は、使用するエンジンを判定するのに十分な情報が得られるまでディレクトリ内のすべてのファイルを調べることになります。

たとえば、次に示すように LIBNAME ステートメントでエンジン名を明示的に指定すると、SAS System は使用するエンジンを判定する必要がなくなります。

```
libname mylib v9 'SAS-library';
```

一部のエンジンでは、そのエンジンを使用するためには、対応するエンジン名を指定する必要があります。たとえば、XML Engine や Metadata Engine を使用するには、各エンジン名と、各エンジンに固有の引数やオプションを指定する必要があります。たとえば、次の LIBNAME ステートメントでは、XML ドキュメントのインポートやエクスポートを行うために、XML Engine を指定しています。

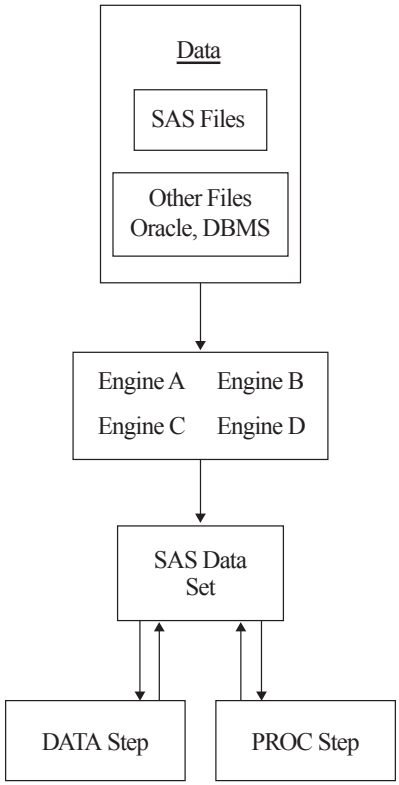
```
libname myxml xml 'C:\MyFiles\XML\MyXmlFile.xml' xmltype=generic;
```

エンジン名は、LIBNAME ステートメント、ENGINE=システムオプション、および**ライブラリの作成**ウィンドウで指定できます。

SAS ファイルとエンジン

次の図は、エンジンを通じて SAS データセットにアクセスする方法を示しています。

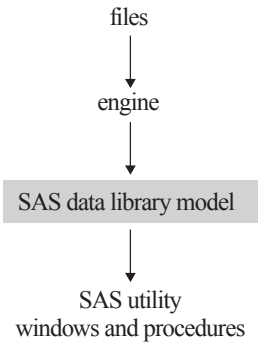
図 35.1 *SAS データセットへのアクセス方法*



- 使用するデータは、SAS System が提供するエンジンによって処理できるファイルの中に格納されています。エンジンは、指定されたデータファイルの位置を確認します。
- エンジンは、データファイルを開き、SAS System が必要とする詳細情報を取得します。この情報には、利用可能な変数、変数の属性、インデックスや圧縮オブザベーションなどの特殊な処理特性の有無、他のエンジンの必要性の有無などがあります。エンジンは、この情報を利用してデータを編成し、SAS System で処理する標準の論理形式にします。
- この標準形式のことを SAS データファイルと呼びます。SAS データファイルは、ディスクリプタ情報と、列(変数)および行(オブザベーション)のテーブル形式に編成されたデータ値から構成されます。
- SAS プロシジャおよび DATA ステップステートメントは、データへのアクセスとデータの処理を、独自の論理形式を使用して行います。エンジンは、物理ファイルの開閉に必要な命令や、適切なフォーマットでデータを読み書きするのに必要な命令を実行します。

エンジンによりアクセスされるデータが SAS データセットモデルへと編成されるのと同様に、エンジンによりアクセスされるファイルのグループは、SAS System で処理するための正しい論理形式へと編成されます。データファイルが SAS ライブラリのメンバとして認識された後は、SAS Utility のウィンドウや SAS プロシジャを使用して、SAS ファイルの内容の一覧表示や管理が行えます。SAS ライブラリの詳細については、24 章, “SAS ライブラリ” (565 ページ) を参照してください。次の図は、エンジンと SAS ライブラリの関係を示しています。

図 35.2 エンジンと SAS ライブラリの関係



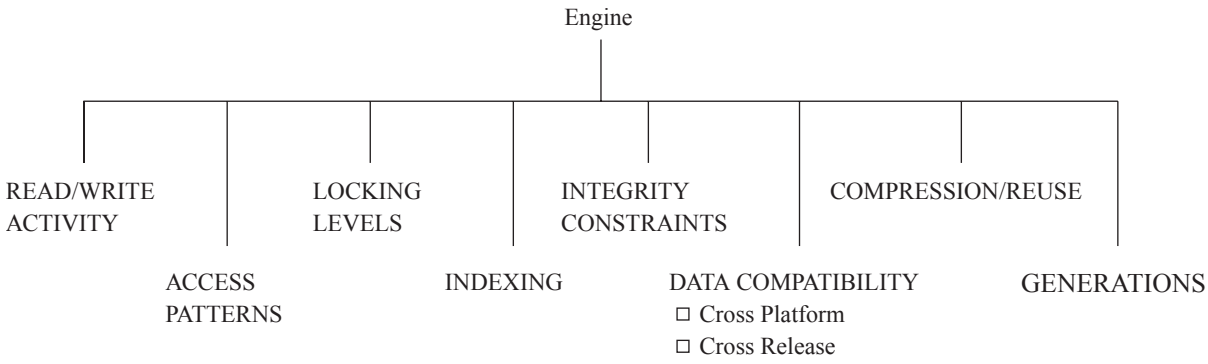
エンジンの特性

エンジンの特性について

SAS データセットのアクセスに使用するエンジンには、それぞれ決まった処理特性があります。さまざまなステートメントやプロシジャは、それぞれ異なる処理特性を必要とします。たとえば、FSEDIT プロシジャでは、選択したデータ値を更新する機能が必要です。さらに、SET ステートメントの POINT=オプションでは、オブザベーションへのランダムアクセス機能と、ファイル内のレコード識別子(RID)を基にオブザベーション番号を計算する機能が必要です。

下記の図に、エンジンが制御する処理を示します。

図 35.3 エンジンが制御する処理



読み取り/書き出し処理

エンジンを使用することにより、次のタスクが行えます。

- SAS データセットに対する読み取り/書き出し処理を、読み取り専用に制限します。
- データセットおよび変数について、属性の更新、削除、名前の変更、再定義を完全にサポートします。
- 前述の機能の一部のみをサポートします。

たとえば、BMDP ファイル、OSIRIS ファイル、SPSS ファイルを処理するエンジンは、読み取り専用の処理をサポートします。SAS ビューを処理するエンジンの中には、SAS プロシジャで既存のオブザベーションの変更を許可するものもあれば、許可しないものもあります。

アクセスパターン

SAS プロシジャや SAS ステートメントを使用して SAS データセット内のオブザベーションを読み取る方法には、次の 4 種類があります。

順次アクセス
: SAS ファイルの先頭からオブザベーションの処理を開始し、ファイル末尾まで順番に処理します。

ランダムアクセス
: インデックスを持つキー変数の値に従ってオブザベーションに直接アクセスして処理します。それより前のオブザベーションは処理しません。

BY グループアクセス
: BY ステートメントで指定された BY 変数の値に基づいてオブザベーションをグループ化し、BY 変数の値の順番に処理します。

マルチパス
: SAS ステートメントまたは SAS プロシジャによって要求されたときに、2 回以上のデータのアクセスを実行します。

SAS ステートメントまたは SAS プロシジャを使用して SAS データセットにアクセスする場合、その SAS データセットを処理するエンジンが、必要なアクセス方法をサポートしていないときは、SAS ログにエラーメッセージが表示されます。

ロックのレベル

SAS System の一部の機能では、データセットが、複数のレベルで更新アクセスをサポートしていることが必要になります。複数の SAS セッションで、または単一セッション内の複数のステートメントやプロシジャで SAS データセットを同時に開くことができる場合、ファイルを同時に読み書きできるセッション、プロシジャ、ステートメントの数は、ロックのレベルによって決まります。たとえば、FSEDIT プロシジャを使用する場合は、1 つのセッション内で同じ SAS データセットのウィンドウを 2 つ開くことができます。エンジンの中には、この機能をサポートしているものと、していないものとがあります。

サポートされているレベルは、レコードレベルとメンバ(データセット)レベルです。メンバレベルのロックでは、複数のセッション、ステートメント、プロシジャを使用して SAS データセットへの読み取りアクセスを行うことができます。ただし、セッション、ステートメント、プロシジャで更新アクセスを行う場合、その SAS データセットに対するその他のアクセスはすべて制限されます。レコードレベルのロックでは、複数のセッション、ステートメント、プロシジャを使用して、SAS データセットへの読み取りアクセスと更新アクセスを同時に行うことができます。ただし、1 つのオブザベーションに対して同時に更新アクセスを行うことはできません。エンジンの中には、これらのレベルの一方をサポートしていないものもあります。

SAS System のデフォルトでは、最大限のレベルの同時アクセスが許可されています。データの一貫性も保証されています。状況によっては、データの一貫性を保証するために、更新アクセスのレベルの制御が必要になることもあります。ロックのレベルを制御するには、CNTLLEV=データセットオプションを使用します。CNTLLEV=データセットオプションを使用すると、次の 3 つのレベルでロックを使用することができます。

- ライブラリ
- データセット
- オブザベーション

CNTLLEV=データセットオプションの使用を検討する必要があるのは、たとえば次のような場合です。

- SAS コンポーネント言語(SCL)アプリケーション、SAS/IML ソフトウェア、DATA ステッププログラムなどのアプリケーションを使用してデータへのアクセスを制御する場合。
- メンバレベルでは制御できないデータに、インターフェイスエンジンを経由して、アクセスする場合。

CNTLLEV=データセットオプションの詳細については、*SAS データセットオプション: リファレンス*を参照してください。

LOCK グローバルステートメントを発行することにより、既存の SAS ファイルに対して排他ロックを設定することもできます。あるファイルに対して排他ロックを設定すると、ロックが解除されるまで、他の SAS セッションはそのファイルに対する読み書きが行えなくなります。LOCK ステートメントに関する詳細については、*SAS Statements: Reference* を参照してください。

注: SAS/ACCESS ソフトウェアや SAS/SHARE ソフトウェアなどの SAS プロダクトは、より強化されたセッション管理サービスとファイルロック機能をサポートするエンジンを搭載しています。

インデックス作成

SAS System の主要な処理の 1 つに、インデックスを持つキー変数の値を使用してオブザベーションに直接アクセスする機能があります。SAS データセットでインデックスを

使用する方法の詳細については、“SAS インデックスについて” (632 ページ) を参照してください。エンジンの中には、インデックス機能をサポートしていないものもあります。

ライブラリエンジンについて

ライブラリエンジンの定義

ライブラリエンジンとは、ファイルのグループにアクセスし、それらを SAS ユーティリティプロシジャやウィンドウで処理可能な論理形式へと変換するエンジンのことです。ライブラリエンジンでは、ライブラリの基本的な処理特性の設定や、ライブラリディレクトリのファイルリストの表示も行います。ライブラリエンジンは、ネイティブライブラリエンジンとインターフェイスライブラリエンジンに分類されます。

ネイティブライブラリエンジン

ネイティブライブラリエンジンの定義

ネイティブライブラリエンジンとは、SAS System に固有の形式のデータファイルのみを読み書きするエンジンのことです。

動作環境の情報
利用できるエンジンは、ホストによって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。また、SAS ソフトウェアプロダクトの中には、下表以外のエンジンを搭載しているものもあります。

デフォルト Base SAS Engine

デフォルト Base SAS Engine は、SAS ライブラリをディスク形式で書き出します。同エンジンは、バージョン 7、バージョン 8、バージョン 9 の SAS ファイルを処理します。新しい SAS ライブラリの作成時に、LIBNAME ステートメントでエンジン名を指定しない場合、自動的にこのエンジンが選択されます。

ディスク上の既存の SAS データセットにアクセスする場合、SAS System はライブラリの内容に基づいてエンジンを割り当てます。Base SAS Engine の特徴は次のとおりです。

- SAS データセットと SAS ライブラリの全機能をエンジンとして唯一サポートしています。
- ビューエンジンをサポートしています。
- SAS ステートメントおよびプロシジャで必要となる処理特性をすべてサポートしています。
- インデックスを作成、管理、使用できます。
- 圧縮された(可変長の)オブザベーションを読み書きします。他のエンジンによって作成された SAS データセットのオブザベーションは、固定長です。
- データセットに永久的なページサイズを割り当てます。また、データセットを処理するときに使用するバッファの数を一時的に割り当てます。
- 破損した SAS データセット、インデックス、カタログを修復します。
- 一貫性制約の適用、バックアップファイルの作成、監査証跡の作成を行います。

注: SAS System のバージョン 7、8、9 で作成された SAS ファイルは、同じファイル形式を持ちます。

Remote Engine

REMOTE Engine は、SAS/SHARE ソフトウェア用の SAS ライブラリエンジンです。同エンジンを使用すると、SAS セッションで SAS Server と通信することにより、共有データにアクセスできるようになります。詳細については、*SAS/SHARE User's Guide* を参照してください。

SASESOCK Engine

SASESOCK Engine は、物理ディスクデバイスに対する入出力ではなく、TCP/IP ポートに対する入出力を処理します。SASESOCK Engine は、パイプ機構により MP CONNECT 処理を実装している SAS/CONNECT アプリケーションで必要となります。詳細については、*SAS/CONNECT User's Guide* を参照してください。

SAS Scalable Performance Data (SPD) Engine

SAS Scalable Performance Data Engine (SPD Engine)は、並列入出力を提供します。また、マルチ CPU を使用して SAS データを読み込み、同データを即座にアプリケーションへと提供できます。SPD Engine では複数のボリュームにまたがるデータを単一のデータセットとして参照できるため、非常に大きなデータセットを処理できます。また、これらのデータセット内のデータを分割し、CPU ごとに実行するマルチスレッド処理で同データを読み込ませることもできます。SPD Engine はデフォルト Base SAS Engine の代わりとなるものではなく、複数ボリュームにまたがらないデータセットの処理には適していません。

このエンジンの機能の詳細については、*SAS Scalable Performance Data Engine: リファレンス*を参照してください。

シーケンシャルエンジン

シーケンシャルエンジンは、ランダムアクセス方式を利用できない記憶媒体にある SAS ファイルを処理します。そのような記憶媒体の例としては、テープや、ディスク上のシーケンシャルフォーマットなどがあります。シーケンシャルエンジンは、デフォルト Base SAS Engine よりも低いオーバーヘッドしか必要としません。これは、シーケンシャルアクセスがランダムアクセスよりも処理が簡単であるためです。ただし、シーケンシャルエンジンは、一部の Base SAS 機能(監査証跡、世代データセット、一貫性制約、インデックス作成)をサポートしていません。

シーケンシャルエンジンは、一部のファイルタイプ(CATALOG、VIEW、MDDB)をバックアップと復元を行うためにのみサポートしています。シーケンシャルエンジンがサポートしていない唯一のファイルタイプは ITEMSTOR です。シーケンシャルエンジンがバックアップと復元以外の目的にも使用できるファイルタイプは DATA のみです。

使用できるシーケンシャルエンジンは次のとおりです。

V9TAPE (TAPE)
　バージョン 7、バージョン 8、バージョン 9 の SAS ファイルを処理します。

V6TAPE
　バージョン 6 の SAS ファイルをバージョン 9 形式に変換することなく処理します。

詳細については、“シーケンシャルデータライブラリ” (577 ページ)を参照してください。

移送エンジン

XPORT Engine は移送ファイルを処理します。このエンジンは、SAS ファイルを、その動作環境に固有の内部表現から、移送ファイルへと変換します。移送ファイルとは、すべてのホストで使用可能なマシン非依存形式のファイルです。移送ファイルを作成するには、LIBNAME ステートメントで XPORT Engine を指定した後、DATA ステップや COPY プロシジャを使用します。

XPORT Engine の使用方法に関する詳細については、*SAS ファイルの移動とアクセス* を参照してください。

V6 互換エンジン

バージョン 6 の SAS ファイルをバージョン 9 形式に変換することなく、バージョン 9 の SAS System で自動的に処理します。

詳細については、33 章, "SAS 9.4 における、以前のリリースの SAS ファイルとの互換性" (717 ページ)、、または **support.sas.com** の Migration Focus Area を参照してください。

インターフェイスライブラリエンジン

インターフェイスライブラリエンジンとは、SAS System 以外のソフトウェアによってフォーマット化されたファイルにアクセスする SAS エンジンです。インターフェイスライブラリエンジンはシステムにより自動的には選択されないため、ユーザーが(LIBNAME ステートメントなどで)明示的に指定する必要があります。

インターフェイスライブラリエンジンには次のものがあります。

SPSS
: SPSS 移送ファイル形式を読み取ります。SPSS 移送ファイル形式は、SAS データセットの移送形式に似た形式です。SPSS 移送ファイル(エクスポートファイルとも呼ぶ)を作成するには、SPSS EXPORT コマンドを使用する必要があります。z/OS 環境では、SPSS Engine は、圧縮フォーマットまたは非圧縮フォーマットの SPSS リリース 9 のファイルや SPSS-X ファイルも読み取ります。

OSIRIS
: EBCDIC 形式の OSIRIS データおよびディクショナリファイルを読み取ります。

BMDP
: BMDP 保存ファイルを読み取ります。

ビューエンジンとは、SAS/ACCESS でサポートされているインターフェイスライブラリエンジンであり、他のベンダのソフトウェアによって作成されたファイルからデータを取得する場合に使用されます。このエンジンを利用することで、DB2 や ORACLE などのデータベース管理システム(DBMS)によって作成されたファイルに対して、データを直接読み書きできるようになります。

ビューエンジンを利用すると、SAS プロシジャや SAS ステートメントを使用して、これらのデータファイルに格納されているデータ値を処理できます。これらのデータファイルを、SAS System によってフォーマットされたファイルに変換および格納するためのリソース使用は発生しません。サイトで利用できる SAS/ACCESS インターフェイスの一覧については、SAS System 管理者に確認してください。SAS/ACCESS ソフトウェアの機能の詳細については、31 章, "SAS/ACCESS ソフトウェアについて" (699 ページ) および使用しているデータベースに対応した SAS/ACCESS ソフトウェアのドキュメントを参照してください。

動作環境の情報
: エンジンの機能およびサポート状況は、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

特殊用途向けエンジン

Character Variable Padding (CVP) Engine

Character Variable Padding (CVP) Engine は、指定された分だけ文字変数の長さを拡張することにより、ファイルでトランスコーディングが必要な場合に文字データの切り捨てが起こらないようにします。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での同じ文字のバイト数と異なる場合に発生します。たとえば、1 バイト文字セット(SBCS)を 2 バイト文字セット(DBCS)やマルチバイト文字セット(MBCS)へとトランスコーディングする場合などがこれに該当します。

CVP Engine は、SAS データファイルに対してのみ使用できる読み取り専用エンジンです。文字変数の長さを拡張するには、次の方法も使用できます。

- CVP Engine を明示的に指定します。たとえば、LIBNAME ステートメントでデフォルトの拡張(変数長を元の 1.5 倍にする)を使用します。
- CVP Engine を暗黙的に指定します。これを行うには、LIBNAME ステートメントで CVPBYTES=または CVPMULTIPLIER=オプションを使用します。これらのオプションには拡張分を指定します。また、CVPENGINE=オプションを使用すると、SAS ファイルの処理に使用するプライマリエンジンを指定できます。デフォルトでは、プライマリエンジンはデフォルト Base SAS Engine になります。

CVP Engine を使用して文字データの切り捨てを防ぐ方法、および LIBNAME ステートメントでの CVP Engine オプションに関する詳細については、*SAS National Language Support (NLS): Reference Guide* を参照してください。

SAS Information Maps LIBNAME Engine

新機能である SAS Information Maps LIBNAME Engine は、SAS Information Map により作成されたデータに読み取り専用でアクセスする方法を提供し、そのデータを SAS セッション内に取り込みます。データを取り出した後は、ほとんどすべての SAS プロシジャを同データに対して実行できるようになります。

Information Map Engine を使用するには、LIBNAME ステートメントでエンジン名として INFOMAPS を指定するとともに、同エンジンに固有の引数やオプションを指定します。

Information Map Engine に関する詳細については、*Base SAS ガイド(Information Map)* を参照してください。

SAS JMP LIBNAME Engine

SAS JMP LIBNAME Engine を使うと、Base SAS セッション内で JMP ファイルの読み書きが行えるようになります。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。JMP で使用する概念や用語については、お使いの JMP パッケージに同梱されているドキュメントを参照してください。

JMP Engine を使用するには、LIBNAME ステートメントでエンジン名として JMP を指定するとともに、同エンジンに固有の引数やオプションを指定します。たとえば、次のコードでは、Baseball.jmp という名前の JMP ファイル内にある 5 つのオブザベーションを読み込んで出力しています。

```
libname b jmp 'C:\JMP\SampleData';

proc print data=b.baseball (obs=5);
run;
```

JMP Engine に関する詳細については、“LIBNAME Statement for the JMP Engine” (*SAS/ACCESS Interface to PC Files: Reference*)を参照してください。

SAS Metadata LIBNAME Engine

このメタデータエンジンは、指定の SAS Metadata Repository に登録されている SAS Metadata Server 上に保存されているメタデータにアクセスします。メタデータとは、データの構造と内容に関する情報であると同時に、データの処理や操作を行うアプリケーションに関する情報でもあります。メタデータには、データの格納場所や、データの処理に使用される SAS Engine などの詳細情報が含まれています。

Metadata Engine は、他の SAS エンジンと同様の方法で使用します。すなわち、まず LIBNAME ステートメントを実行してライブラリ参照名を割り当て、エンジン名を指定します。それ以降は、そのライブラリ参照名が有効である SAS セッション全体を通じて、同ライブラリ参照名を使用できます。ただし、メタデータのライブラリ参照名は、SAS ライブラリの物理的な格納場所に関連付けられるのではなく、SAS Metadata Server 上の特定のリポジトリ内に保存されている指定のメタデータオブジェクトに関連付けられます。メタデータオブジェクトは、SAS ライブラリとそのメンバの処理に必要となる SAS エンジンおよびオプションを定義します。

Metadata Engine を指定する LIBNAME ステートメントを実行すると、その Metadata Engine は、ターゲット SAS ライブラリに関する情報を対応するメタデータから取り出します。Metadata Engine は、この情報を使用して、エンジンを指定する LIBNAME ステートメントを構築し、同ステートメントに適切なオプションを割り当てます。その後、ユーザーのデータにアクセスする必要がある場合、この Metadata Engine は、基盤となるエンジンを使用して同データを処理します。

Metadata Engine を呼び出すには、LIBNAME ステートメントや**ライブラリの作成**ウィンドウ内で、エンジン名 META をその Metadata Engine の固有の引数とオプションとともに明示的に指定します。

Metadata Engine の使用方法に関する詳細については、*SAS Language Interfaces to Metadata* を参照してください。

SAS XML LIBNAME Engine

SAS XML LIBNAME Engine を使うと、XML ドキュメントを SAS データセットとしてインポートすることや、SAS データセットを XML ドキュメントとしてエクスポートすることができます。

- このエンジンは、XML マークアップを SAS System 独自のフォーマットに変換することにより、外部 XML ドキュメントをインポートします(すなわち、入力ファイルとして読み取ります)。
- また、このエンジンは、SAS System 独自のフォーマットを XML マークアップへと変換することにより、SAS データセットを XML ドキュメントへとエクスポートします(すなわち、出力ファイルへと書き出します)。

XML Engine を使用するには、エンジンニックネーム **XML** または **XMLV2** のいずれかを(たとえば、LIBNAME ステートメント内、または**ライブラリの作成**ウィンドウで)特定の引数やオプションと一緒に指定します。

XML Engine の使用方法に関する詳細については、*SAS XML LIBNAME Engine: ユーザーガイド*を参照してください。

36 章

SAS のファイル管理

SAS アプリケーションのパフォーマンスの向上

SAS System には、メモリなどのコンピュータリソース使用量を管理してパフォーマンスを調整するツールが用意されています。これらの機能を使用しなくても、ほとんどの SAS アプリケーションは、使用している動作環境で効率よく実行されます。ただし、次の状況でのアプリケーション開発では、パフォーマンスの調整が必要な場合があります。

- 大きなデータセットを操作する場合
- 定型的な処理を繰り返し実行する場合
- データセンタ用のパフォーマンスのガイドラインを確立する場合
- 大きな SAS データセットに対して SAS/FSP ソフトウェアなどを使用して、対話型クエリを実行する場合

パフォーマンスの向上に関する詳細については、12 章, “システムパフォーマンスの最適化” (183 ページ)を参照してください。

動作環境間での SAS ファイルの移動

SAS ファイルを異なる動作環境間で移動する手順は、使用する動作環境、移動する SAS ファイルのメンバタイプやバージョン、および使用可能なファイルの移動方法によって異なります。

詳細については、*SAS ファイルの移動とアクセス*を参照してください。

破損した SAS ファイルの修復

SAS ファイルの破損の検出

Base SAS Engine は、インデックス、一貫性制約、監査証跡ファイルを含む SAS データファイルと SAS カタログに発生する破損を検出します。また、一部の破損については、復元する手段を提供します。SAS ファイルの更新中に次のいずれかの事象が発生した場合、SAS System は、ファイルの破損を修復し、一部の破損については復元します。

- データファイルまたはカタログの更新中にシステム障害が発生した場合。
- データファイル、インデックスファイル、監査証跡ファイル、カタログが格納されるディスクの空き領域がなく、ファイルの書き出しが完了しなかった場合。
- データファイル、インデックスファイル、監査証跡ファイル、カタログへの書き出し中に、入出力エラーが発生した場合。

障害が発生すると、データファイルまたはカタログに書き出されていないオブザベーションやレコードが失われ、値が格納された場所についての情報に問題が生じます。そのファイルが次回読み込まれたときに、SAS System は、ファイルに破損があることを認識し、データセットが切り詰められていないならば、DLDMGACTION=データセットオプションまたはシステムオプションの設定に従ってファイルを最大限に修復します。切り詰められているデータセットを修復するには、REPAIR ステートメントを使用します。

データファイルが置かれているストレージデバイスが破損した場合、バックアップデバイスを使用して、破損したデータファイル、インデックス、監査ファイルを修復できます。

注: SAS ビュー(DATA ステップビュー、PROC SQL ビュー、SAS/ACCESS ビュー)や、コンパイル済みストアド DATA ステッププログラムを修復することはできません。メンバタイプが VIEW または PROGRAM である SAS ファイルが破損した場合は、ファイルを再作成する必要があります。

注: SAS データファイルの監査証跡ファイルが破損した場合は、監査証跡を終了するまでデータファイルを処理できません。この場合、新しい監査証跡ファイルを作成するか、監査証跡ファイルを使用しないでデータファイルを処理することができます。

SAS データファイルの修復

破損した SAS データファイルを開く場合の処理の方法を指定するには、DLDMGACTION=データセットオプションまたはシステムオプションを設定します。データファイルの破損が検出されると、SAS System は次の指定に従って自動的に対応します。

DLDMGACTION=FAIL
確認メッセージのプロンプトを出さずにステップを中止し、要求されたファイルの破損を示すエラーメッセージを SAS ログに出力するよう SAS に指示します。この指定により、アプリケーションで修復の判断を制御することや問題の発生を検出することができます。

破損したデータファイルを回復するには、DATASETS プロシジャの REPAIR ステートメントを発行します。詳細については、*Base SAS Procedures Guide* を参照してください。

DLDMGACTION=ABORT
ステップを中止し、要求ファイルの破損を示すエラーメッセージを SAS ログに出力して、SAS セッションを終了するよう SAS に指示します。

DLDMGACTION=REPAIR
自動的にファイルを修復し、インデックス、一貫性制約、監査証跡ファイルを再作成します。修復が正常に終了した場合は、正常にファイルが開かれて修復されたことを示すメッセージを SAS ログに出力します。修復が正常に終了しない場合は、確認メッセージを表示しないで処理が中止し、ファイルの破損を示すエラーメッセージが SAS ログに出力されます。

注: データファイルが大きいと、修復時間は長くなります。

DLDMGACTION=NOINDEX
SAS System がデータファイルの修復、インデックスと一貫性制約の無効化、インデックスファイルの削除、データファイルを更新して無効化されたインデックスと一貫性制約を反映すること、データファイルを INPUT モードのみでオープンするよう制限することなどを自動的に行うよう指示します。無効化されたインデックスと一貫性制約を修正し、インデックスファイルを再構築するには DATASETS プロシジャの REBUILD ステートメントを実行するように指示する警告メッセージが SAS ログに書き出されます。詳細については、“無効化されたインデックスと一貫性制約の修復” (751 ページ)を参照してください。

DLDMGACTION=PROMPT
対話型モードとバッチモードの両方で、バージョン 6 と同じ動作を行います。対話型モードでは、FAIL、ABORT、REPAIR 処理を選択するためのダイアログボックスが表示されます。バッチモードでは、ファイルは開かれません。

データファイルでは、最終修復日時と、修復回数の合計が自動的に保持されます。破損ログを表示するには、次の CONTENTS プロシジャを使用します。

```
proc contents data="c:\temp\testuser\large";
run;
```

アウトプット 36.1 CONTENTS プロシジャ出力

The SAS System

The CONTENTS Procedure

Data Set Name	TESTDATA.LARGE	Observations	10000
Member Type	DATA	Variables	2
Engine	V9	Indexes	0
Created	Wed, Dec 22, 2010 04:34:42 PM	Observation Length	112
Last Modified	Wed, Dec 22, 2010 04:35:24 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	12288
Number of Data Set Pages	92
First Data Page	1
Max Obs per Page	109
Obs in First Data Page	98
Number of Data Set Repairs	5
Last Repair	16:35 Wednesday, December 22, 2010
Filename	c:\temp\TestUser\large.sas7bdat
Release Created	9.0301B0
Host Created	NET_ASRV

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	filler	Char	100
2	i	Num	8

インデックスの修復

前述の障害に加えて、SAS データファイルのインデックスが破損することがあります。動作環境のコマンドを使用して、SAS データファイルの削除、コピー、名前変更のいずれかを実行した際に、関連するインデックスファイルを処理しなかった場合は、そのインデックスが破損する可能性があります。インデックスは、SAS データファイルと同様に、DLDMGACTION=データセットオプションまたはシステムオプションを使用して修復します。また、DATASETS プロシジャの REPAIR ステートメントを使用して、破損した複合インデックスおよび単一インデックスを再構築できます。

次のいずれかの操作によって削除されたインデックスは、REPAIR ステートメントを使用して修復することはできません。

- COPY プロシジャまたは DATASETS プロシジャ以外の方法、たとえば、DATA ステップを使用して SAS データファイルをコピーした場合。
- SORT プロシジャの FORCE オプションを使用して、元のデータファイルを上書きした場合。

これらの場合には、DATASETS プロシジャの INDEX CREATE ステートメントを使用して、明示的にインデックスを再構築する必要があります。

無効化されたインデックスと一貫性制約の修復

DLDMGACTION=NOINDEX データセットオプションまたはシステムオプションを使用している場合に、破損したデータファイルを SAS System が検出すると、SAS System は次のことを行います。

- インデックスや一貫性制約を使用せずにデータファイルを自動的に修復します。
- インデックスと一貫性制約を無効化します。
- インデックスファイルを削除します。
- データファイルを更新し、無効化されたインデックスと一貫性制約を反映します。
- データファイルを INPUT モードのみでオープンするよう制限します。
- 次の警告をログに書き出します。

```
WARNING:SAS data file MYLIB.MYFILE.DATA was damaged and has been partially repaired.To c
```

データファイルは、PROC DATASETS の REBUILD ステートメントが実行されるまで、INPUT モードのままになります。このステートメントを使用して、インデックスと一貫性制約を修復し、インデックスファイルを再構築するか、それとも無効化されたインデックスと一貫性制約を削除するかを指定します。詳細については、*Base SAS Procedures Guide* に記載されている PROC DATASETS の REBUILD ステートメントを参照してください。

カタログの修復

破損した SAS カタログを開こうとする際に SAS System がどのような処置を行うかを指定するには、DLDMGACTION=データセットオプションまたはシステムオプションを設定します。SAS System は、カタログの破損を検出すると、指定した処理方法に従って自動的に対応します。

注: カタログの破損には、次の 2 種類があります。

- 部分的な破損は、ディスクの状態に起因するものです。この破損により、メモリ内のデータがディスクに書き出されない場合があります。更新のために開かれ

ているカタログエントリは、破損したものとして認識されます。破損したエントリは 1 つずつ検査され、すべてのレコードをエラーなく読み込むことが可能か判別されます。

- 重大な破損は、重大な入出力エラーに起因するものです。カタログ全体が破損したものとして認識されます。

DLDMGACTION=FAIL
確認メッセージを表示しないで処理を中止し、ファイルの破損を示すエラーメッセージを SAS ログに表示します。この指定により、アプリケーションで修復の判断を制御することや問題の発生を検出することができます。

破損したカタログを回復するには、DATASETS プロシジャの REPAIR ステートメントを発行します。詳細については、*Base SAS Procedures Guide* を参照してください。REPAIR ステートメントを使用してカタログを復元すると、破損した可能性があるエントリについての警告が表示されます。復元されたエントリに含まれる更新内容は、破損が発生する以前にディスクに書き出されたものに限られる可能性があります。

DLDMGACTION=ABORT
ステップを中止し、ファイルの破損を示すエラーメッセージを SAS ログに表示して、SAS セッションを中止します。

DLDMGACTION=REPAIR
部分的な破損の場合は、カタログを自動的に検査してどのエントリが破損したかを確認します。エントリの読み込み中にエラーが発生すると、そのエントリがコピーされます。コピー処理でエラーが発生すると、そのエントリは自動的に削除されます。重大な破損の場合は、カタログ全体が新しいカタログにコピーされます。

DLDMGACTION=PROMPT
部分的な破損の場合は、対話型モードとバッチモードの両方で、バージョン 6 と同じ動作をします。対話型モードでは、FAIL、ABORT、REPAIR 処理を選択するためのダイアログボックスが表示されます。バッチモードでは、ファイルは開かれません。重大な破損の場合は、カタログ全体が新しいカタログにコピーされます。

データファイルとは異なり、カタログでは破損に関するログは保持されません。

37 章

外部ファイル

外部ファイルの定義

外部ファイル

外部ファイルとは、SAS System ではなくオペレーティングシステムによって管理されるファイルです。外部ファイルには、SAS System 固有の形式を持たないテキストやデータが格納されています。外部ファイルは、SAS カタログまたは出力デバイスである場合もあります。SAS ジョブは、外部ファイルとして SAS ログを作成します。SAS ジョブで作成される外部ファイルは、通常、プロシジャ出力の形式または DATA ステップ出力の形式で作成されます。

SAS セッションで使用する外部ファイルには、SAS ジョブへの入力として次のものがあります。

- DATA ステップへの入力として使用する生データ
- サブミットする実行用 SAS プログラムステートメント

また、外部ファイルには、SAS ジョブからの出力として次のものがあります。

- SAS ログ(SAS ジョブの記録)
- DATA ステップによって作成されるレポート
- SAS プロシジャによって作成されるプロシジャ出力。標準のリスト出力、バージョン 8 以降での Output Delivery System (ODS)からの HTML 出力および PostScript 出力など

また、PRINTTO プロシジャにより、プロシジャ出力を外部ファイルに送信できます。詳細については、“PRINTTO” (*Base SAS Procedures Guide*)を参照してください。ODS の詳細については、9 章, “SAS 出力” (145 ページ) を参照してください。

注: データベース管理システム(DBMS)のファイルは特殊なファイルです。DBMS ファイルは、SAS/ACCESS ソフトウェアを使用して読み込むことができます。DBMS ファイルの詳細については、31 章, “SAS/ACCESS ソフトウェアについて” (699 ページ) および各 DBMS 用の SAS/ACCESS ソフトウェアのドキュメントを参照してください。

動作環境の情報
外部ファイルを SAS ジョブで使用するには、動作環境に固有の情報が必要です。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

直接的な外部ファイルの参照

SAS ステートメントまたは SAS コマンドで外部ファイルを直接的に参照するには、ファイルの物理名を引用符で囲んで指定します。ファイルの物理名とは動作環境がファイルを識別する名前です。次の表に例を示します。

表 37.1 *直接的な外部ファイルの参照*

外部ファイルに関するタスク	手段	例

入力データソースとして外部ファイルを指定する場合	INFILE	`data weight;`
		`  infile 'input-file';`
		`  input idno $ week1 week16;`
		`  loss=week1-week16;`

外部ファイルを書き出す指定をする場合	FILE	`file 'output-file ';`
		`if loss ge 5 and loss le 9 then`
		`  put idno loss 'AWARD STATUS=3';`
		`  else if loss ge 10 and loss le 14 then`
		`  put idno loss 'AWARD STATUS=2';`
		`  else if loss ge 15 then`
		`  put idno loss 'AWARD STATUS=1';`
		`run;`

別の外部ファイルから生データを読み込む場合	%INCLUDE	`%include 'source-file';`

間接的な外部ファイルの参照

プログラムの一部で外部ファイルを参照していて、別の SAS ジョブや後続の処理で参照先を容易に変更できるようにするには、ファイル参照名を使用して、外部ファイルを間接的に参照するようにします。FILENAME ステートメント、FILENAME 関数、適切なオペレーティングシステムのコマンドを使用して、ファイル参照名やニックネームを外

部ファイルに割り当てます。[1] ファイル参照名は、外部ファイルである SAS カタログ、または出力デバイスに割り当てることができます。次の表に例を示します。

表 37.2 *間接的な外部ファイルの参照*

外部ファイルに関するタスク	手段	例
入力データソースとして外部ファイルにファイル参照名を割り当てる場合	FILENAME	`filename mydata 'input-file';`
出力データを格納する外部ファイルにファイル参照名を割り当てる場合	FILENAME	`filename myreport 'output-file';`
プログラムステートメントを含む外部ファイルにファイル参照名を割り当てる場合	FILENAME	`filename mypgm 'source-file';`
出力デバイスにファイル参照名を割り当てる場合	FILENAME	`filename myprinter <device-type>` `  <host-options>;`

入力データソースとして外部ファイルを指定する場合	INFILE	`data weight;`
		`  infile mydata;`
		`  input idno $ week1 week16;`
		`  loss=week1-week16;`

外部ファイルを書き出す指定をする場合	FILE	`file myreport;`
		`if loss ge 5 and loss le 9 then`
		`  put idno loss 'AWARD STATUS=3';`
		`  else if loss ge 10 and loss le 14 then`
		`  put idno loss 'AWARD STATUS=2';`
		`  else if loss ge 15 then`
		`  put idno loss 'AWARD STATUS=1';`
		`run;`

別の外部ファイルから生データを読み込む場合	%INCLUDE	`%include mypgm;`

1 動作環境によっては、コマンド'&'を使用してファイル参照名を割り当てることもできます。

複数の外部ファイルの効率的な参照

ディレクトリや区分データセット(PDS または MACLIB)のような単一の集約記憶域に含まれている複数のファイルを使用する場合、単一のファイル参照名に続けてファイル名をかっこで囲んで指定することにより、個々のファイルにアクセスできます。ファイル参照名を使用することにより、ファイルの格納場所の長い名前を繰り返し入力する必要はなくなります。また、ファイルの格納場所を後で変更する場合にも、プログラムの変更が簡単です。ファイル参照名をディレクトリに割り当てる例を、次の表に示します。

表 37.3 *複数ファイルの効率的な参照*

外部ファイルに関するタスク	手段	例
ディレクトリにファイル参照名を割り当てる場合	FILENAME	`filename mydir 'directory-or-PDS-name';`

入力データソースとして外部ファイルを指定する場合	INFILE	`data weight;`
		`  infile mydir(qrt1.data);`
		`  input idno $ week1 week16;`
		`  loss=week1-week16;`

外部ファイルを書き出す指定をする場合 *	FILE	`file mydir(awards);`
		`if loss ge 5 then put idno loss`
		`  'AWARD STATUS=3';`
		`  else if loss ge 10`
		`  then put idno loss 'AWARD STATUS=2';`
		`  else if loss ge 15`
		`  then put idno loss 'AWARD STATUS=1';`
		`run;`

別の外部ファイルから生データを読み込む場合	%INCLUDE	`%include mydir(whole.program);`

* SAS System によって、動作環境で適切な拡張子を持つファイルが作成されます。

その他のアクセス方式での外部ファイルの参照

外部ファイルにファイル参照名を割り当てる場合、次の FILENAME アクセス方式を使用して外部ファイルにアクセスできます。

- CATALOG
- DATAURL
- FTP
- Hadoop
- SFTP
- TCP/IP SOCKET

- URL
- WebDAV
- ZIP

それぞれのアクセス方式の使用方法について、次の表に例を示します。

表 37.4 その他のアクセス方式での外部ファイルの参照

外部ファイルに関するタスク	手段	例
SAS カタログにファイル参照名を割り当てる場合	FILENAME ステートメント CATALOG 指定子	`filename mycat catalog 'catalog' <catalog-options>;`
データ URL を使用してアクセスする外部ファイルにファイル参照名を割り当てる場合	FILENAME ステートメント DATAURL 指定子	`filename myfile dataurl 'external-file' <dataurl-options>;`
FTP を使用してアクセスする外部ファイルにファイル参照名を割り当てる場合	FILENAME ステートメント FTP 指定子	`filename myfile FTP 'external-file' <ftp-options>;`
Hadoop 分散ファイルシステムを通じてアクセスする外部ファイルにファイル参照名を割り当てる場合	FILENAME ステートメント Hadoop 指定子	`filename myfile hadoop 'external-file' <hadoop-options>;`
SFTP を使用してアクセスする外部ファイルにファイル参照名を割り当てる場合	FILENAME ステートメント SFTP 指定子	`filename myfile SFTP 'external-file' <sftp-options>;`

TCP/IP ソケットをクライアントモードまたはサーバーモードで使用してアクセスする外部ファイルにファイル参照名を割り当てる場合	FILENAME ステートメント SOCKET 指定子	`filename myfile SOCKET 'hostname: portno' <tcpip-options>;`
		または
		`filename myfile SOCKET ':portno' SERVER <tcpip-options>;`

外部ファイルに関するタスク	手段	例
URL を使用してアクセスする外部ファイルにファイル参照名を割り当てる場合	FILENAME ステートメント URL 指定子	`filename myfile URL 'external-file' <url-options>;`
WebDAV サーバーを通じてアクセスする外部ファイルにファイル参照名を割り当てる場合	FILENAME ステートメント WebDAV 指定子	`filename myfile WEBDAV 'external-file' <webdav-options>;`
Zlib サービスを使用してアクセスする ZIP ファイルにファイル参照名を割り当てる場合	FILENAME ステートメント ZIP 指定子	`filename myfile ZIP 'external-file' <zip-options>;`

各ステーメントに関する詳細については、*SAS Statements: Reference* を参照してください。

外部ファイルの操作

外部ファイルの読み込み

通常、生データから SAS データセットを作成する際に、外部ファイルを SAS ジョブに読み込みます。詳細については、19 章, “生データの読み込み” (417 ページ)を参照してください。

外部ファイルへの書き出し

外部ファイルへの書き出しには、次の方法を使用します。

- SAS DATA ステップ
- 外部ファイルインターフェイス(EFI)
- エクスポートウィザード

DATA ステップを使用して、カスタマイズされたレポートを作成する場合は、外部ファイルにレポートを出力します。レポートを出力する最も単純な DATA ステップは、次のようになります。

```
data _null_;
   set budget;
   file 'your-file-name';
   put variables-and-text;
run;
```

DATA ステップによるレポートの作成例については、18 章, “DATA ステップの処理” (389 ページ)を参照してください。

動作環境がグラフィカルユーザーインターフェイスをサポートしている場合は、EFI またはエクスポートウィザードを使用して外部ファイルに書き出すことができます。EFI は、ポイントアンドクリックのグラフィカルインターフェイスで、SAS 内部形式ではないデータの読み書きに使用できます。EFI は、外部ファイルの SAS データセットへの読み込みや、SAS データセットの外部ファイルへの書き出しに使用できます。EFI の詳細については、*SAS/ACCESS Interface to PC Files: Reference* を参照してください。

注: EFI に渡すデータファイルがパスワードで保護されている場合、ログイン ID とパスワードを何回も入力するよう求められます。

エクスポートウィザードは、SAS データセットからデータを読み込んで、外部ファイルに書き出す操作を行います。エクスポートウィザードは、ウィザード形式のアプリケーションで SAS System の一部として機能します。ウィザード上に選択肢を表示することにより、データの処理を支援します。ウィザードの詳細については *SAS/ACCESS Interface to PC Files: Reference* を参照してください。

外部ファイルの処理

外部ファイルからデータを読み込む場合や、外部ファイルにデータを書き出す場合、DATA ステップを使用して次の操作も行えます。

- 各レコードの一部分のみを別のファイルにコピーする。
- ファイルをコピーし、各レコードにフィールドを追加する。
- 1 つの DATA ステップで複数のファイルを同じ方法で処理する。
- ファイルのサブセットを作成する。
- 外部ファイルを同じ場所で更新する。
- 異なるコンピュータ環境で読み込まれるファイルにデータを書き出す。
- ファイル内部のエラーをビットレベルで修正する。

DATA ステップを使用して外部ファイルを処理する例については、19 章, “生データの読み込み” (417 ページ)を参照してください。

5 部

SAS の対応する標準的なプロトコル

38 章

SMTP 電子メールインターフェイス

SMTP を経由した電子メールの送信

SMTP (Simple Mail Transfer Protocol)電子メールインターフェイスを使用して、SAS からプログラム経由で電子メールを送信できます。SMTP は、SAS System を実行しているすべての動作環境で利用できます。SAS System の電子メールサポート機能を使用して SMTP に対応した電子メールを送信するには、SMTP をサポートしているイントラネットまたはインターネット接続環境が必要となります。

一部の SMTP サーバーでは、ログイン ID として使用されるユーザー ID のみが必要となります。それ以外のサーバーでは、完全な電子メールアドレスが必要となります。SAS System の SMTP 電子メールインターフェイスは、ユーザー ID を次の順番で認証します。

1. ユーザー ID が EMAILHOST=システムオプションの USERID=オプションに指定されている場合、SAS SMTP 電子メールインターフェイスはこのユーザー ID を使用して認証しようとします。
2. ユーザー ID が USERID=オプションに指定されていない場合、SAS SMTP 電子メールインターフェイスは FILENAME=ステートメントの FROM=オプションに指定されたユーザー ID を使用して認証しようとします。
3. ユーザー ID が FILENAME=ステートメントの FROM=オプションに指定されていない場合、SAS SMTP 電子メールインターフェイスは EMAILID=システムオプションに指定されたユーザー ID を使用して認証しようとします。
4. ユーザー ID が EMAILID=システムオプションに指定されていない場合、SAS SMTP 電子メールインターフェイスは、オペレーティングシステムからユーザー ID を検索し、そのユーザー ID を認証しようとします。

SAS からの電子メール送信の詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

SMTP 電子メールを制御するシステムオプション

SAS System では、SMTP に対応した電子メールを制御するシステムオプションがいくつか提供されています。動作環境および SMTP 電子メールインターフェイスがサイトでサポートされているかどうかに応じて、これらのオプションを起動時に指定するか、または SAS 設定ファイル内に指定する必要があります。

動作環境の情報
動作環境におけるデフォルトの電子メールインターフェイスの判定方法や、システムオプションを設定するための正しい構文の決定方法については、使用している動作環境に対応する SAS ドキュメントを参照してください。

EMAILSYS システムオプションは、SAS System 内部からの電子メール送信に使用する電子メールシステムを指定します。EMAILSYS システムオプションの詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

次に示すシステムオプションは、お使いのサイトで SMTP 電子メールインターフェイスがサポートされている場合にのみ指定できます。

EMAILACKWAIT=
SMTP サーバーから肯定応答を受け取るために、SAS が待機する秒数を指定します。詳細については、“EMAILACKWAIT= System Option” (*SAS System Options: Reference*)を参照してください。

EMAILAUTHPROTOCOL=
SMTP 対応の電子メールで使用する認証プロトコルを指定します。詳細については、“EMAILAUTHPROTOCOL= System Option” (*SAS System Options: Reference*)を参照してください。

EMAILFROM
FILE または FILENAME ステートメントを使用して電子メールを送信する場合に、FROM=オプションが必要となるかどうかを指定します。詳細については、“EMAILFROM System Option” (*SAS System Options: Reference*)を参照してください。

EMAILHOST
お使いのサイトで電子メール機能をサポートしている SMTP サーバーを指定します。詳細については、“EMAILHOST= System Option” (*SAS System Options: Reference*)を参照してください。

EMAILPORT
SMTP サーバーに接続するためのポートを指定します。詳細については、“EMAILPORT System Option” (*SAS System Options: Reference*)を参照してください。

EMAILUTCOFFSET
電子メールメッセージの Date:ヘッダーフィールドで使用する UTC オフセットを指定します。詳細については、“EMAILUTCOFFSET= System Option” (*SAS System Options: Reference*)を参照してください。

次に示すシステムオプションは、SMTP 以外の電子メールシステムでも指定できます。

EMAILID=
SAS System の内部から電子メールを送信する個人の ID を指定します。詳細については、“EMAILID= System Option” (*SAS System Options: Reference*)を参照してください。

EMAILPW=
電子メール用のログインパスワードを指定します。詳細については、“EMAILPW= System Option” (*SAS System Options: Reference*)を参照してください。

SMTP 電子メールを制御するステートメント

FILENAME ステートメント

FILENAME ステートメントで EMAIL (SMTP)アクセスメソッドを指定すると、SMTP 電子メールインターフェイスを使用して SAS プログラムから電子メールを送信できます。詳細については、“FILENAME Statement” (*SAS Statements: Reference*)を参照してください。

FILE ステートメントと PUT ステートメント

FILE ステートメント内には各種の電子メールオプションを指定できます。FILE ステートメント内に指定した電子メールオプションは、FILENAME ステートメント内に指定した対応する電子メールオプションよりも優先されます。

DATA ステップで、FILE ステートメントを使用して電子メールファイル参照名を出力先として定義した後、PUT ステートメントを使用してメッセージ本文を定義します。PUT ステートメントの各種ディレクティブは、FILE ステートメントおよび FILENAME ステートメントに指定された電子メールオプションよりも優先されます。

39 章

汎用一意識別子(UUID)

汎用一意識別子と Object Spawner

汎用一意識別子について

汎用一意識別子(UUID)とは、日時情報、およびホストの IEEE ノードアドレスから構成される 128 ビット長の識別子です。UUID は、行のような SAS Application の構成要素を一意に識別する必要がある場合に使用します。たとえば、SAS System がサーバーとして稼働しており、複数のクライアントに対してオブジェクトを同時に配布している場合、各オブジェクトを UUID に関連付けられます。これにより、確実に、特定のクライアントと SAS System が同一のオブジェクトを参照できます。

Object Spawner について

Object Spawner とは、サーバー上で要求を待ち受けるプログラムのことです。Object Spawner は要求を受け取ると、接続を受け付け、接続が行われたポートまたはサービスに関連付けられているアクションを実行します。Object Spawner は、UUID ジェネレータデーモン(UUIDGEND)となるように設定できます。UUIDGEND は、要求を行うプログラム向けに UUID を生成します。現時点では、SAS System は Windows 環境でのみ UUID を生成できます。UUIDGEND は、UUID の生成をネイティブサポートしていないホスト上で実行されている SAS セッション向けに UUID を生成します。

UUID ジェネレータデーモンの定義

UUIDGEND の定義は、Object Spawner の呼び出し時にユーザーが指定する設定ファイルに記述します。この構成ファイルで、UUID 要求を待機するポートを識別し、Windows 以外の動作環境ではこの構成ファイルで UUID ノードも特定します。

Windows 以外の動作環境に UUIDGEND をインストールする場合は、SAS テクニカルサポート(http://support.sas.com/techsup/contact/)に連絡して、UUID ノードを取得してください。UUIDGEND が本当に一意の UUID を生成するためには、UUID ノードは、インストールされた UUIDGEND システムごとに一意でなければなりません。

Windows 以外の動作環境における UUIDGEND 設定ファイルの例を次に示します。

```
#
##  Define our UUID Generator Daemon.  Since this UUIDGEND is
##  executing on a UNIX host, we contacted SAS Technical
##  Support to get the specified sasUUIDNode.
#
dn: sasSpawnercn=UUIDGEND,sascomponent=sasServer,cn=SAS,o=ABC Inc,c=US
objectClass: sasSpawner
sasSpawnercn: UUIDGEND
sasDomainName: unx.abc.com
sasMachineDNSName: medium.unx.abc.com
sasOperatorPassword: myPassword
sasOperatorPort: 6340
sasUUIDNode: 0123456789ab
sasUUIDPort: 6341
description: SAS Session UUID Generator Daemon on UNIX
```

Windows 環境における UUIDGEND 設定ファイルの例を次に示します。

```
#
##  Define our UUID Generator Daemon.  Since this UUIDGEND is
##  executing in a Windows operating environment, we do not need to specify
##  the sasUUIDNode.
#
dn: sasSpawnercn=UUIDGEND,sascomponent=sasServer,cn=SAS,o=ABC Inc,
c=US
objectClass: sasSpawner
sasSpawnercn: UUIDGEND
sasDomainName: wnt.abc.com
sasMachineDNSName: little.wnt.abc.com
sasOperatorPassword: myPassword
sasOperatorPort: 6340
sasUUIDPort: 6341
description: SAS Session UUID Generator Daemon on XP
```

UUID ジェネレータデーモンのインストール

設定ファイルを作成した後、UUIDGEND をインストールするには、この設定ファイルを指定して Object Spawner プログラム(objspawn)を起動します。使用する構文は次のとおりです。

objspawn -configFile *filename*
configFile オプションの省略形として-cf を使用できます。

filename には、UUIDGEND 設定ファイルの完全修飾パス名を指定します。パス名に空白が含まれている場合、同パス名を一重引用符または二重引用符で囲みます。Windows 環境では、パス名に空白が含まれている場合、同パス名を二重引用符で囲みます。z/OS 環境では、この設定ファイルを次のように指定します。

//dsn:*myid.objspawn*`.log` (MVS ファイルの場合)

//hfs:*filename*.`ext` (OpenEdition ファイルの場合)

Windows 環境では、`objspawn.exe` ファイルは、***`SAS-installation-directory`*`\SASFoundation\`*`SAS-version`*`\`ディレクトリにインストールされます。たとえば、典型的な Windows のインストールでは、`objspawn.exe` ファイルは次のディレクトリにインストールされます。

```
C:\Program Files\SASHome\SASFoundation\9.4
```

UNIX 環境では、objspawn ファイルは、SAS System のインストールディレクトリ内の `utilities/bin` ディレクトリにインストールされます。

VMS 環境では、OBJSPAWN_STARTUP.COM ファイルは、分離されたプロセスとして OBJSPAWN.COM ファイルを実行します。OBJSPAWN.COM ファイルは Object Spawner を実行します。OBJSPAWN.COM ファイルには、お使いのサイトで適切なバージョンの Object Spawner の実行、ディスプレイノードの設定、テンプレート DCL ファイル(OBJSPAWN_TEMPLATE.COM)を指すプロセスレベルの論理名の定義、Object Spawner の起動前に必要となるアクションの実行などに必要となるその他のコマンドも含まれています。

OBJSPAWN_TEMPLATE.COM ファイルは、クライアントの実行に必要となる設定を実行します。Object Spawner は、まず論理名 SAS$OBJSPAWN_TEMPLATE が定義されているかどうかを確認します。定義されている場合、同テンプレートファイル内のコマンドが、クライアントセッションの起動時に使用されるコマンドシーケンスの一部として実行されます。この論理名を定義する必要はありません。

SAS 言語要素での UUID の割り当て

SAS 言語要素での UUID の割り当ての概要

Windows 以外のプラットフォーム上で SAS アプリケーションを実行する場合、UUIDGEND をインストールしているならば、次の方法を使用して UUID を割り当てることができます。

- UUIDGEN 関数
- UUIDCOUNT=システムオプション
- UUIDGENDHOST システムオプション

UUIDGEN 関数

UUIDGEN 関数は、各セルの UUID を返します。詳細については、“UUIDGEN Function” (*SAS Functions and CALL Routines: Reference*)を参照してください。

UUIDCOUNT=システムオプション

UUIDCOUNT=システムオプションには、UUID ジェネレータデーモンを使用するたびに取得する UUID の数を指定します。詳細については、“UUIDCOUNT= System Option” (*SAS System Options: Reference*)を参照してください。

UUIDGENDHOST システムオプション

UUIDGENDHOST システムオプションには、動作環境と UUID ジェネレータデーモンのポートを指定します。詳細については、“UUIDGENDHOST= System Option” (*SAS System Options: Reference*)を参照してください。

40 章

Internet Protocol Version 6 (IPv6)

IPv6 の概要

SAS 9.2 では、次世代インターネットプロトコルである IPv6 を導入しました。IPv6 は、現在広く使用されているインターネットプロトコルである IPv4 の後継規格です。SAS は、IPv4 を IPv6 で置き換えるのではなく、両方のプロトコルをサポートしています。IPv4 から IPv6 への完全な移行までには長い時間がかかることが予想されるため、その移行期間中は、これら 2 種類のプロトコルが共存することになります。

新しいプロトコルへの移行が必要な第一の理由としては、32 ビット長の IPv4 アドレス空間がすでに枯渇していることが挙げられます。IPv6 は、128 ビット長のアドレススキームを使用します。このアドレススキームは、IPv4 よりもはるかに多くの IP アドレスを提供します。

IPv6 が IPv4 よりも有利である点としては、次のことが挙げられます。

- より大きなアドレス空間を提供(32 ビット長ではなく 128 ビット長)
- ヘッダーフォーマットが簡略化されている
- 自動設定が可能
- より効率的なルーティングが可能
- サービス品質およびセキュリティが改善されている
- 各種の規制要件に準拠している
- グローバルマーケットでの広範な使用が可能

IPv6 アドレス形式

IPv6 と IPv4 では、異なるアドレス形式を使用します。各プロトコルのアドレス形式を比較したものを下記の表に示します。

表 40.1 *IPv6 と IPv4 のアドレス形式の比較*

特徴	IPv6	IPv4
アドレス空間	128 ビット	32 ビット
データ表現	文字列	整数
バイト長(フィールドセパレータを含む)	39	15
フィールドセパレータ	コロン(:)	ピリオド(.)
表記法	16 進	10 進
IP アドレスの例	db8:0:0:1	10.23.2.3

IPv6 アドレスの例

IPv6 アドレスの完全表記と短縮表記の例

IPv6 アドレスの完全表記の例を次に示します。

```
FE80:0000:0000:0000:0202:B3FF:FE1E:8329
```

完全表記では、128 ビットのアドレスを 8 つの 16 ビットブロックに区切り、*global:subnet:interface* 形式で表示します。

IPv6 アドレスの省略表記の例を次に示します。

```
FE80::0202:B3FF:FE1E:8329
```

::(連続する 2 つのコロン)表記は、連続する 16 ビットブロックがすべてゼロを含んでいることを表す場合に使用されます。上記の例ではゼロを含む連続する 4 つのブロックを::で置き換えています。SAS ソフトウェアは省略表記の IP アドレスを検出すると、同アドレスを 8 つの 16 ビットブロックからなる完全表記の 128 ビットアドレス形式へと再構築します。

ポート番号を含む IPv6 アドレスの例

ポート番号を含んでいる IP アドレスの例を次に示します。

```
[2001:db8:0::1]:80
```

大かっこは、ポート番号を指定する場合にのみ必要となります。大かっこは、アドレスをポート番号から分離するために使用されます。ポート番号を使用しない場合、大かっこを省略できます。

省略表記では、ゼロを含んでいる連続するブロックを::で置き換えることができます。たとえば次のように記述します。

```
[2001:db8::1]:80
```

URL を含む IPv6 アドレスの例

URL を含んでいる IP アドレスの例を次に示します。

```
http://[2001:db8:0::1]:80
```

接頭辞 **http://**は、続く文字列が URL であることを示します。大かっこは、ポート番号を指定する場合にのみ必要となります。大かっこは、アドレスをポート番号から分離するために使用されます。ポート番号を使用しない場合、大かっこを省略できます。

完全修飾ドメイン名(FQDN)

IP アドレスは容易に変更されるため、ハードコーディングされた IP アドレスを含んでいる SAS アプリケーションでは、保守上の問題が発生しやすくなります。

このような問題を避けるために、IP アドレスのかわりに FQDN を使用することを推奨します。FQDN に関連付けられている IP アドレスを取り出すのは、TCP/IP プロトコルの一部である名前解決システムの役目になります。

次のプログラム例は、停止されたリポジトリ内のクライアントアクティビティを修復するものです。

```
PROC METAOPERATE
   SERVER="d6292.us.company.com"
   PORT=2222
   USERID="myuserid"
   PASSWORD="mypassword"
   PROTOCOL=BRIDGE

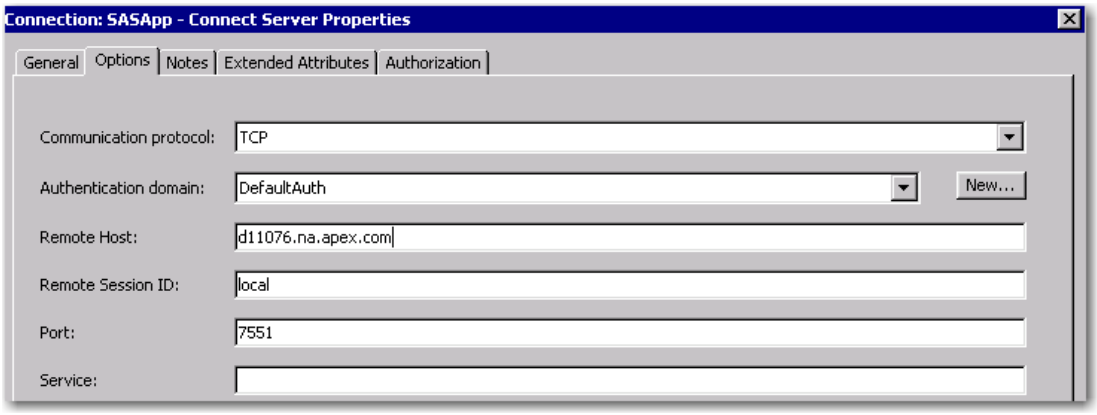
   ACTION=RESUME
    OPTIONS=""
   NOAUTOPAUSE;
```

上記のプログラムで IP アドレスを使用した場合、コンピュータノード名に関連付けられていた IP アドレスが変更されると、同プログラムは正しく動作しなくなります。

FQDN を使用すれば、それに割り当てられている IP アドレスが変更された場合であっても、元のプログラムのままで問題なく動作します。TCP/IP の名前解決システムは、FQDN を、現在それに割り当てられている IP アドレスへ自動的に解決します。

SAS System の GUI アプリケーションで FQDN を指定する例を次に示します。

図 40.1 SAS 管理コンソールのウィンドウで FQDN を指定する例



d11076.na.apex.com という FQDN を、SAS 管理コンソールの **CONNECT Server プロパティ**ウィンドウ内にある**リモートホスト**フィールドに指定しています。

一部の SAS プロダクトでは、ホスト名の長さに制限が適用される場合があります。

次の例では、SAS メニュー変数に FQDN を割り当てています。

```
%let sashost=hrmach1.dorg.com;
rsubmit sashost.sasport;
```

この FQDN は 8 文字より長いため、SAS マクロ変数に割り当てる必要があります。この SAS マクロ変数が RSUBMIT ステートメントで使用されています。

推奨資料

- *Learning SAS by Example:A Programmer's Guide*
- *The Little SAS Book:A Primer*
- *Output Delivery System:The Basics and Beyond*
- *Base SAS Procedures Guide*
- *Base SAS Utilities: リファレンス*

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

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

用語集

1 対 1 のマージ
MERGE ステートメント(BY ステートメントなし)を使用し、データセットにおけるオブザベーションの位置に基づいて、2 つ以上のデータセットのオブザベーションを結合する処理。

1 対 1 のマッチング
複数の SET ステートメントを使用して各データセットからオブザベーションを読み込むことによって、複数のデータセットにあるオブザベーションを結合し、1 つのオブザベーションにする処理。

1 次元配列
同じ種類の変数を 1 つの名前でグループ化すること。処理されると、この変数グループ化の結果は単純な列形式で示されます。

2 次元配列
2 つの次元を使用して、同じデータ型の変数を 1 つの名前にグループ化すること。処理されると、この変数グループ化の結果は単純な列形式で示されます。

Application Response Measurement
業界のパートナーシップによって開発され、ソフトウェアアプリケーションの可用性およびパフォーマンスのモニタに使用されるアプリケーションプログラミングインターフェイスの名前。ARM では、特定のビジネスにとって重要なアプリケーションタスクがモニタされます。

ARM
Application Response Measurement を参照。

ARM エージェント
ソフトウェアベンダによる ARM API の実装。各 ARM エージェントは、アプリケーションで呼び出せる実行可能ルーチンのセットです。ARM エージェントは SAS と同時に実行されます。SAS アプリケーションからトランザクション情報が渡されると、ARM エージェントは ARM トランザクションレコードを収集し、ARM ログに書き出します。

ARM サブシステム
PROC および DATA ステップ処理ならびにファイル入出力処理などの、内部 SAS 処理トランザクションのグループ。1 つのサブシステムかまたはすべてのサブシステムをオンにするには、ARM システムオプション ARMSUBSYS=を使用します。

ARM システムオプション
SAS ARM インターフェイスのさまざまな側面を制御する SAS システムオプションのグループ。

ARM マクロ
アプリケーションの応答時間を測定するマクロ。ARM マクロは、ARM API 関数呼び出しを起動します。適切なマクロパラメータとマクロ変数を設定することで、条件付き実行が許可されます。ARM マクロは、SAS マクロ機能には含まれていません。

ARM ログ
ARM トランザクションのレコードを含む外部ファイル。

ASCII 照合順序
テキストデータを並べ替えるために特定の ASCII エンコーディングで使用される規則。並べ替え順序は、ASCII エンコーディングのコードページにおける各コードポイントの場所によって決定されます。Windows Latin1 コードページでは、優先並べ替え順序は、区切り文字、数字、大文字、小文字です。大文字 A (コードポイント 41)は小文字 g (コードポイント 67)より先に来るため、A は g より前に並べ替えられます。

autoexec ファイル
SAS の起動時に自動で実行される SAS ステートメントを含むファイル。autoexec ファイルを使用すると、一部の SAS システムオプションを指定したり、頻繁に使用されるデータソースにライブラリ参照名およびファイル参照名を割り当てたりできます。

BMDP Engine
BMDP ファイルへの読み取り専用アクセスを提供する SAS エンジン。BMDP は、もともと UCLA 健康科学コンピュータ施設(UCLA Health Sciences Computing Facility)で開発された統計分析プログラムのライブラリです。

BY 値
BY 変数の値。

BY グループ
BY ステートメントで指定された変数に対して、同じ値を有するオブザベーションまたは行のグループ。BY ステートメントで複数の変数が指定されている場合、BY グループはそれらの変数の値の固有な組み合わせを持つオブザベーションの集まりになります。

BY グループ処理
オブザベーションを 1 つ以上の変数の値に従って並べ替え、グループ化またはインデックスを付けるために BY ステートメントを使用する処理。BY グループ処理は多数の SAS プロシジャと DATA ステップによってサポートされています。たとえば、PRINT プロシジャで BY グループ処理を使用すると、1 つの SAS データセット内の異なるオブザベーショングループに対して別々のレポートを印刷できます。

BY グループ変数
BY 変数を参照。

BY 変数
BY ステートメントで指定された変数。処理されるオブザベーションのグループがその値により定義されます。

CALL ルーチン
SAS CALL ルーチンを参照。

catref
カタログ参照名(catalog reference)を参照。

CEDA
クロス環境データアクセスを参照。

CPU
中央演算処理装置を参照。

CPU 時間
コンピュータシステムの中央演算処理装置が、要求された計算またはその他の操作を実行するのにかかる時間。

DATA ステップ
SAS プログラムで、DATA ステートメントで始まり、RUN ステートメント、別の DATA ステートメント、PROC ステートメント、ジョブ終了のいずれかで終わるステートメントのグループ。DATA ステップでは、生データまたはその他 SAS データセットの読み取り、SAS データセットの作成ができます。

DATA ステップインターフェイス
DATA ステップを使用して SAS のその他コンポーネント(マクロ機能や Output Delivery System (ODS)など)と対話できるようにする SAS の機能。

DATA ステップビュー
ストアド DATA ステッププログラムから成る SAS データセットの一種。DATA ステップビューには、他の場所に格納されたデータの定義が含まれています。ビューに物理データは含まれません。ビューの入力データは、外部ファイルおよびその他の SAS データセットを含む 1 つ以上のソースから取得できます。DATA ステップビューはその他のファイルを読み取る(入力のために開く)のみなので、ビューのソースデータは更新できません。

DBMS
データベース管理システムを参照。

DCB
データ制御ブロックを参照。

DICTIONARY テーブル
現在の SAS セッションで使用中または使用可能な SAS データライブラリ、SAS データセット、SAS マクロ、外部ファイルに関する情報を提供する、多数の読み取り専用 SAS データビュー。DICTIONARY テーブルには、現在有効な SAS システムオプションの設定も含まれています。

DOCUMENT 出力先
出力オブジェクトの階層を生成する SAS Output Delivery System (ODS)出力先。DOCUMENT 出力先を使用すると、PROC ステップまたは DATA ステップを再実行しなくても複数の ODS 出力形式を表示でき、出力構造に対するユーザーの制御が強化されます。

DO グループ
単純な DO ステートメントで始まり、対応する END ステートメントで終わる一連のステートメント。

DO ループ
反復 DO、DO WHILE または DO UNTIL ステートメントで始まり、対応する END ステートメントで終わる一連のステートメント。ステートメントは、DO ステートメントで指定された指示に従って(通常は繰り返し)実行されます。

DSD
区切り文字を区別するデータを参照。

EBCDIC 照合順序
テキストデータを並べ替えるために特定の EBCDIC エンコーディングで使用される規則。並べ替え順序は、EBCDIC エンコーディングのコードページにおける各コードポイントの場所によって決定されます。たとえば、ドイツ語の EBCDIC コードページでは、優先並べ替え順序は、区切り記号、数字、大文字、小文字です。

FIRST 変数
各 BY グループの最初のオブザベーションを識別するために作成される一時変数。この変数は、SAS データセットに追加されません。

FTP
ファイル転送プロトコルを参照。

GB
ギガバイトを参照。

HTML
ハイパーテキストマークアップ言語を参照。

I/O 時間
入出力時間を参照。

I/O バウンドアプリケーション
処理対象データの配信可能速度によってパフォーマンスに制約を受けるアプリケーション。複数 CPU、I/O の分割、スレッド技術、RAID (redundant array of independent disks)技術、またはこれらの組み合わせによって、問題を緩和できます。

ISO
国際標準化機構(International Organization for Standardization)を参照。

KB
2 の 10 乗すなわち 1024 バイト。

KEYS エントリ
対話型ウィンドウプロシジャのファンクションキー設定を含む SAS カタログエントリの一種。

L10N
ローカライズを参照。

LAST 変数
各 BY グループの最後のオブザベーションを識別するために作成される一時変数。この変数は、SAS データセットに追加されません。

LISTING 出力
モノスペースフォントの SAS プロシジャ出力。リスト出力のテキストはすべてフォントサイズが同じで、特別なフォントスタイルは適用されません。

LISTING 出力先
従来の SAS 出力(モノスペース形式)を生成する ODS 出力先。

MB
2 の 20 乗すなわち 1,048,576 (およそ 100 万)バイト。

MDDB
多次元データベースを参照。

Metadata LIBNAME Engine
メタデータで識別されるデータの処理および補強を行う SAS エンジン。メタデータエンジンは、指定したメタデータリポジトリのメタデータオブジェクトから、ターゲット SAS ライブラリについての情報を取得します。

MultiVendor Architecture
SAS システムの基になるストラテジ。この場合、コードのおよそ 70%が任意のホスト(アプリケーション層)で移植可能(すなわち再利用可能)です。残りのコードは、コア層とホストコード層に分けられ、SAS がリリースされるプラットフォームそれぞれで大幅に書き直されます。SAS System を使用して開発された MVA SAS アプリケーションはホスト非依存であり、実行するためには SAS システムがインストールされている必要があります。

ODS
Output Delivery System を参照。

ODS 出力
任意の ODS 出力先によって生成されたフォーマット出力。たとえば、OUTPUT 出力先では SAS データセット、LISTING 出力先ではリスト出力、HTML 出力先ではハイパーテキストマークアップ言語でフォーマットされた出力が生成されます。

ODS 出力先
ODS (Output Delivery System)が特定の種類の出力を生成するために使用する出力先。ODS 出力先の種類には、HTML、XML、リスト、PostScript、RTF、SAS データセットが含まれますが、これに限定されるわけではありません。

ODS スタイル
SAS 出力の外観を提供するために使用される、色、フォント、線、マーカー記号などの組み合わせ。ODS でスタイルを定義するには、スタイルテンプレートを使用します。

Output Delivery System
マークアップ言語(HTML、XML)、PDF、リスト、RTF、PostScript、SAS データセットなどのさまざまな出力形式で出力を生成できる SAS のコンポーネント。

PCL
Printer Command Language を参照。

PDV
プログラムデータベクトルを参照。

PF キー
ファンクションキーを参照。

PMENU エントリ
PMENU プロシジャで作成されたプルダウンメニュー、メニューバーおよびダイアログボックスの定義を含むカタログエントリの一種。

Printer Command Language
Hewlett-Packard プリンタを制御するために Hewlett-Packard で開発されたコマンド言語。各 PCL コマンドは、ESC をキーとしてその後に続く一連のコード番号から成ります。さまざまなモデルや種類の Hewlett-Packard プリンタで使用するために、さまざまなバージョンの PCL が開発されました。

PROC SQL ビュー
SQL プロシジャによって作成される SAS データセット。PROC SQL ビューにはデータは含まれません。かわりに、そこに格納されている情報を使用して、SAS データファイル、SAS/ACCESS ビュー、DATA ステップビュー、それ以外の PROC SQL ビューなどのその他のファイルからデータ値を読み取ることができます。PROC SQL ビューの出力は、1 つ以上のファイルのサブセットかスーパーセットのどちらかです。

PROC ステップ
SAS プロシジャを呼び出して実行する SAS ステートメントのグループ。PROC ステップでは、通常、入力として SAS データセットが使用されます。

RMF
リソース測定機能を参照。

RTF 出力先
Microsoft Word 2000 で使用されるリッチテキスト形式で書き出された出力を生成する ODS 出力先。

SAS CALL ルーチン
SAS 言語要素の一種。1 つ以上の引数を処理するために使用し、式で使用できる結果が返されます。

SAS/ACCESS ビュー
ファイルの一種。他のソフトウェアベンダのファイル形式で格納されたファイルからデータ値を取得します。SAS/ACCESS ビューを作成するには、SAS/ACCESS ソフトウェアの ACCESS プロシジャを使用します。

SAS/SHARE サーバー
SERVER プロシジャの実行結果であり、SAS/SHARE ソフトウェアの一部です。サーバーは個別の SAS セッションで実行され、1 つ以上の SAS ライブラリへの入出力要求を制御/実行して、ユーザーの SAS セッションにサービスを提供します。

Sashelp ライブラリ
SAS ソフトウェアが提供する SAS ライブラリ。ヘルプウィンドウ、デフォルトのファンクションキー定義およびウィンドウ定義、メニューのテキストが格納されています。

SASHELP ライブラリ
Sashelp ライブラリを参照。

sasroot
サイトまたはコンピュータで SAS がインストールされるディレクトリまたはフォルダの名前表記。

Sasuser.Profile カタログ
特定のユーザーまたはサイトに対する SAS ウィンドウ環境の属性についての情報が格納される SAS カタログ。これには、ファンクションキー定義、グラフィックアプリケーションのフォント、ウィンドウ属性、対話型 SAS プロシジャで使用されるその他の情報が含まれます。

Sasuser ライブラリ
最初の SAS セッションの開始時に作成されるデフォルトの永久 SAS ライブラリ。Sasuser ライブラリには、SAS 用に指定したカスタマイズ済み機能または設定を格納する PROFILE カタログが含まれます。

SASUSER ライブラリ
Sasuser ライブラリを参照。

SAS 演算子
SAS 式で、比較、論理演算または算術計算を要求するいくつかの記号のすべて。

SAS エンジン
エンジンを参照。

SAS カタログ
さまざまな種類の情報を、カタログエントリと呼ばれるより小さな単位で格納する SAS ファイル。単一の SAS カタログに、異なる種類のカタログエントリを含めることができます。

SAS カタログエントリ
SAS カタログ内での個々の格納単位。各エントリには使用目的に応じて、さまざまなエントリタイプが割り当てられています。

SAS 日付値
SAS で日付を表す整数。この整数は、1960 年 1 月 1 日から指定の日付までの日数を表します。たとえば、SAS 日付値 366 はカレンダ日付 1961 年 1 月 1 日を表します。

SAS 日付定数
SAS ステートメントで日付を表す'ddMMMyy'd または'ddMMMyyyy'd 形式の文字列。文字列は引用符で囲み、その後に文字 d を続けます('6JUL01'd、'06JUL01'd、'6 JUL2001'd、'06JUL2001'd など)。

SAS キーワード
SAS 言語の主要部分となるリテラル。たとえば、SAS キーワードには、DATA、PROC、RUN、SAS 言語要素名、SAS ステートメントオプション名、システム変数などが含まれます。

SAS コマンド
SAS を起動するコマンド。このコマンドは、動作環境とサイトによって異なる場合があります。

SAS 時間値
SAS で時間を表す整数。この整数は、現在日付の午前 0 時から指定の時間値までの秒数を表します。たとえば、午前 9 時 30 分の SAS 時間値は 34200 です。

SAS 時間定数
SAS ステートメントで時間を表す'hh:mm:ss't 形式の文字列。文字列は引用符で囲み、その後に文字 t を続けます('09:53:22't など)。

SAS 式
数値、文字値またはブール値を生成するために評価される命令セットを形成する、一連のオペランドと算術演算子から成るマクロ式の一種。オペランドの例としては、定数やシステム関数があります。SAS はプログラムステートメントで算術演算式を使用して、変数の作成、値の割り当て、新しい値の計算、変数の変換、条件付き処理を行います。

SAS システムオプション
SAS 言語要素の一種。SAS セッション中、多数の操作に適用されます。システムオプションでは、SAS セッションの初期化、ハードウェアとソフトウェアとの SAS 相互作用、および SAS ファイルの入出力処理を制御できます。

SAS 初期化
SAS セッションを開始するために有効にする必要があるグローバル特性の設定処理。初期化オプションとよばれる特定の SAS システムオプションを設定して、初期化を行います。SAS 初期化は、SAS の起動時に自動的に行われます。

SAS ステートメント
SAS 言語要素の一種。SAS プログラムで特定の操作を実行したり、SAS プログラムに情報を提供したりするために使用されます。

SAS データセット
いずれかのネイティブ SAS ファイル形式の内容を含むファイル。SAS データセットには、SAS データファイルと SAS データビューです。

SAS データセットオプション
特定の SAS データセットに対して適用するアクションを指定する SAS 言語要素。たとえば、データセットオプションを使用すると、変数の名前変更、処理するオブザベーションの選択、パスワードの指定などが行えます。

SAS データビュー
他のファイルからデータ値を取得する SAS データセットの一種。SAS データビューに含まれているのは、変数(列)のデータ型やデータ長などのディスクリプタ情報のみです。これに加えて、他の SAS データセットや、他のソフトウェアベンダのファイル形式で格納されているファイルからデータ値を取得するのに必要となる他の情報などがあります。

SAS 名
SAS 変数や SAS データセットなどの項目に割り当てられた名前。ほとんどの SAS 名では、最初の文字はアルファベットかアンダースコアにする必要があります。以降の文字には、アルファベット、数字またはアンダースコアを使用できます。ブランクと特殊文字は使用できません(アンダースコアを除く)。ただし、SAS 変数名に適用する規則は、VALIDVARNAME=システムオプションで確定されます。SAS 名の最大長は、その SAS 名が割り当てられる言語要素によって異なります。

SAS 日時値
SAS で日付と時間を表す整数。この整数は、1960 年 1 月 1 日午前 0 時から指定の日時までの秒数を表します。たとえば、2000 年 6 月 5 日午前 9 時 30 分の SAS 日時値は 1275816600 です。

SAS 日時定数
SAS で日付と時間を表す'ddMMMyy:hh:mm:ss'dt または 'ddMMMyyyy:hh:mm:ss'dt 形式の文字列。文字列は引用符で囲み、その後に文字 dt を続けます('06JUL2001:09:53:22'dt など)。

SAS ファイル
SAS が作成、編成、管理する特殊構造化ファイル。SAS ファイルには、SAS データセット、カタログ、ストアドプログラム、アクセスディスクリプタ、ユーティリティファイル、多次元データベースファイル、財務データベースファイル、データマイニングデータベースファイル、アイテムストアファイルなどがあります。

SAS プログラム
入力データの読み込みや変換、出力の生成を行うため、1 つまたは一連のプロセスで SAS に指示する SAS ステートメントのグループ。DATA ステップとプロシジャステップ(単独かまたは組み合わせて使用)が SAS プログラムの基礎となります。

SAS プロシジャ
レポート生成、ファイル管理、データ分析などの特定タスクを実行するための内蔵プログラムを参照する SAS 言語要素の一種。

SAS 変数
SAS データセット内または SAS データビュー内の列。各変数のデータ値は、すべてのオブザベーション(行)の単一の特性を表します。

SAS 命名規則
SAS 変数、データセット、ライブラリ参照名、ファイル参照名、その他の SAS プログラミング言語構成要素に対する有効な名前の構成を記述する規則。

SAS レジストリ
1 つ以上の SAS ソフトウェアプロダクトの構成データを含むアイテムストア。たとえば、SAS レジストリには、SAS の起動時に割り当てられるデータライブラリとファイルショートカット、SAS エクスプローラポップアップメニューのメニュー定義、および定義されているプリンタについての情報が格納されています。通常、SAS レジストリは、REGEDIT コマンドで起動されるレジストリエディタで表示されます。

SAS ログ
入力した SAS ステートメントの記録、およびプログラムの実行についてのメッセージを含むファイル。

Scalable Performance Data Engine
SAS エンジン。このエンジンはデータを編成して効率化されたファイル形式にするため、アプリケーションにデータを迅速に提供できます。

SMF
システム管理機能を参照。

SMP
対称型マルチプロセッシングを参照。

SOURCE エントリ
カタログエントリの一種。SAS テキストエディタウィンドウからのテキストが含まれます。

SPD Engine
Scalable Performance Data Engine を参照。

SQL
構造化照会言語を参照。

TB
2 の 40 乗すなわち 1,099,511,627,776 (およそ 1.1 兆)バイト。

TCP/IP
ネットワークプロトコルのペアの省略名。伝送制御プロトコル(TCP)は、イーサネットなどのローカルエリアネットワークで情報を転送するための標準プロトコルです。TCP によって、プロセス間情報が適切な順序で配信されます。インターネットプロトコル(IP)は、動作環境間の接続を管理するためのプロトコルです。IP はネットワー

クを介して情報を特定の動作環境に送信し、転送情報のフラグメント化と再構成を行います。

Unicode
ほぼ世界中の書記体系で使用されている文字および記号の交換、処理、表示をサポートするための業界標準となっている、16 ビットのエンコーディング。

User ライブラリ
ライブラリ参照名 User が割り当てられた SAS ライブラリ。ライブラリ参照名 User が定義されると、1 レベル名が付いた SAS データセットは、一時ライブラリ Work ではなくこのライブラリに格納されます。

WHERE 句
後ろに 1 つ以上の WHERE 式が続くキーワード WHERE から構成される構文文字列です。WHERE 句は、データセット内のオブザベーションを選択するために使用される条件を定義します。

WHERE 式
オブザベーションの選択条件を定義する WHERE 句内の構文文字列です。たとえば、メンバシップに関するデータベースで、"WHERE member_type=Senior"を指定すると、すべてのシニアメンバが返されます。

WHERE 処理
WHERE 式の実行時に使用され、処理するオブザベーションを条件に従って選択する方法。

Work ライブラリ
各 SAS セッションまたは SAS ジョブの最初に、SAS が自動的に定義する一時 SAS ライブラリ。User ライブラリが指定されていない場合、新たに作成される 1 レベル名の SAS ファイルは、デフォルトでは Work ライブラリに置かれ、現在の SAS セッションまたはジョブの最後に削除されます。

WORK ライブラリ
Work ライブラリを参照。

XML
拡張可能マークアップ言語を参照。

XMLMap ファイル
SAS XML LIBNAME エンジンに XML ドキュメントの解釈方法を指示する XML タグを含むファイル。

XPORT Engine
SAS 移送エンジン。このエンジンによって移送形式の SAS ファイルにアクセスします。

アイテムストア
さまざまな情報から成る SAS ライブラリメンバ。情報には個々にアクセス可能です。アイテムストアのコンテンツは、UNIX System Services または Windows で使用されるディレクトリ構造に似たディレクトリツリー構造で編成されています。たとえば、特定の値を、ディレクトリパス(root_dir/sub_dir/value)を使用して格納/検索できます。SAS レジストリはアイテムストアの一例です。

アクセスディスクリプタ
SAS、データベース管理システムまたは PC ベースのソフトウェアアプリケーション (Microsoft Excel、Lotus 1-2-3、dBASE など)によって管理されるデータが記述され

る SAS/ACCESS ファイル。アクセスディスクリプタは作成後に、1 つ以上のビューディスクリプタを作成するための基礎として使用できます。

アクセス方式
ファイルアクセス方式を参照。

暗号化
データを変換して目的の受取人以外は判読できない形式にする作業や処理。

移送エンジン
SAS ファイルを、その動作環境に固有の内部表現から移送形式に変換する機能。

移送形式
SAS データセット、SAS データライブラリおよび SAS カタログを、ある動作環境から別の動作環境に移動するために使用される 2 つのファイル形式のうちのどちらか。1 つの移送形式は、COPY プロシジャで XPORT Engine が使用されるときに生成されます。もう 1 つの移送形式は、CPORT および CIMPORT プロシジャによって生成されます。これらの移送形式はそれぞれ、すべての動作環境において同一です。

移送ファイル
移送形式の SAS ライブラリ、SAS カタログまたは SAS データセットを含むシーケンシャルファイル。移送ファイルを使用すると、SAS データライブラリ、SAS カタログおよび SAS データセットを、ある動作環境から別の動作環境に移動できます。

一時 SAS データセット
現在のプログラムまたは対話型 SAS セッションの間のみ存在するデータセット。一時 SAS データセットは、それ以降の SAS セッションでは使用できません。

一時的な配列要素
動作は変数に似ているが出力データセットには表示されない配列要素。一時的な配列要素は名前を持たず、配列名と次元のみで参照できます。それらは自動的に保持され、次の DATA ステップ反復が開始されても欠損値にリセットはされません。

位置パラメータ
マクロパラメータの一種。%MACRO ステートメントの起動時に(カンマ区切り文字を使用して)指定され、(再度カンマ区切り文字を使用して)マクロ実行ステートメントの対応する位置に定義されます。

一貫性制約
SAS データファイルの変数に格納できる有効なデータ値を制限するデータ検証規則。一貫性制約を指定すると、格納データの有効性と整合性を管理できます。

印刷ファイル
キャリッジコントロール(プリンタコントロール)情報を含む外部ファイル。

インターフェイスビュー
SAS/ACCESS ソフトウェアを使用して作成される SAS データビューです。インターフェイスビューを使用すると、データベース管理システム(DBMS)や PC データファイル内に格納されているデータの読み書きが行えます。インターフェイスビューは、DATA ステップやプロシジャに対する入力として使用されます。

インターフェイスビューエンジン
SAS/ACCESS ソフトウェアで使用される SAS エンジンの一種。別ベンダのソフトウェアでフォーマットされたファイルからデータを取得するために使用します。

SAS/ACCESS インターフェイスごとに固有のインターフェイスビューエンジンがあり、そのエンジンはインターフェイス製品データを読み取って、SAS が理解可能な形式(すなわち SAS データセット)でデータを返します。

インターフェイスライブラリエンジン
他社のソフトウェアによりフォーマット化されたファイルにアクセスする場合に使用される SAS ライブラリエンジンです。たとえば、インターフェイスライブラリエンジンは、SPSS、OSIRIS、BMDP などのデータにアクセスする場合に使用されます。インターフェイスビューエンジンは、SAS/ACCESS ソフトで使用されるインターフェイスライブラリエンジンの一種です。

インデックス
SAS データセットのオブザベーションに SAS が迅速かつ効率的にアクセスできるようにする SAS データセットのコンポーネント。SAS インデックスの目的は、WHERE 句処理を最適化し、BY グループ処理を促進することです。

生データ
統計分析において、標準化などの特定の操作が実行されていないデータ(SAS データセットのデータを含む)。

生データファイル
レコードのフィールドにデータ値が含まれている外部ファイル。DATA ステップでは、INFILE ステートメントおよび INPUT ステートメントを使用して生データファイルを読み込みます。

永久 SAS データセット
現在のプログラムまたは対話型 SAS セッションが終了した後も削除されない SAS データセット。永久 SAS データセットは、現在以降の SAS セッションで使用可能です。

永久 SAS ファイル
SAS セッションまたはジョブの終了時に削除されない SAS ライブラリ内のファイル。

永久 SAS ライブラリ
SAS セッションの終了時に削除されないため、それ以降の SAS セッションで使用可能な SAS ライブラリ。

演算子
SAS 演算子を参照。

エンジン
ファイルからの読み取りやファイルへの書き出しを行う SAS のコンポーネント。さまざまなエンジンによって、異なる種類のファイル形式にアクセスできます。

エントリの種類
カタログエントリの構造と属性を SAS で特定する SAS カタログエントリの特性。SAS カタログエントリを作成すると、自動的にエントリの種類が名前の一部に割り当てられます。

オーバーヘッド
ベンチマーキングで、プログラムがコンポーネントをはじめて使用する際、SAS ソフトウェアのそのコンポーネント(プロシジャなど)をメインメモリに移動するために使用されるリソースの追加分。

オープンコード
いずれのマクロ定義にも属さない SAS プログラムの一部。

オブザベーション
SAS データセットの行。オブザベーション内のデータ値はすべて、顧客や状態などの単一のエンティティに関連付けられます。各オブザベーションには、各変数に対する 1 つのデータ値か欠損値インジケータのどちらかが含まれます。

オペランド
演算子、変数および定数を含む式における変数や定数。

外部キーの一貫性制約
外部キーデータファイル内にある 1 つ以上の変数のデータ値を、主キーデータファイル内にある対応する変数や値へリンクする、参照一貫性制約。

外部ファイル
ホストオペレーティングシステムまたは別のベンダのソフトウェアアプリケーションによって作成および管理されるファイル。外部ファイルは、データと格納された SAS ステートメントの両方が読み取り可能です。

拡張可能マークアップ言語
コンテンツ、意味または使用法に関するタグを付けて情報を構造化するマークアップ言語。構造化情報には、コンテンツ(ワードや数字など)とコンテンツが果たす役割の指示の両方が含まれます。たとえば、セクション見出しのコンテンツは、データベーステーブルのコンテンツとは異なる意味があります。

拡張属性
オブジェクトの標準メタデータに含まれないカスタム属性。拡張属性を使用すると、タスクを自動化できます。この場合、カスタム属性を 1 つ以上のオブジェクトに関連付ける必要があります。拡張属性は、プログラムか手動(アプリケーションウィンドウを使用)のどちらかで追加できます。

仮数
数の表現部分。指数表記 Ew.d 出力形式とコンピュータ上の浮動小数点プロセッサはどちらも仮数、指数および底によって数を表現します。このとき、数=仮数 * (底**指数)です。

仮数(significand)
仮数(mantissa)を参照。

カタログ
SAS カタログを参照。

カタログエントリ
SAS カタログエントリを参照。

カタログ参照名(catalog reference)
一時的にカタログまたは連結カタログに関連付けられる名前。カタログ参照名を割り当てるには、CATNAME ステートメントを使用します。

日付値
SAS 日付値を参照。

日付間隔
経過した時間を日または月でカウントするための単位。

日付スタンプ
コンピュータによって記録されるトランザクションの日付。例としては、SAS ログ開始時の日付または出力ファイル、SAS セッションの初期化、SAS ファイルの作成または変更で出力された日付などが挙げられます。

日付定数
SAS 日付定数を参照。

日付と時間出力形式
数値を日付、時間および日時として書き出す方法を SAS に指示する命令。

日付と時間入力形式
日付、時間および日時として表された数値を読み取る方法を SAS に指示する命令。

カラム出力
DATA ステップにおける出力スタイルで、固定カラムにデータを書き出すために PUT ステートメントでカラム番号を指定します。

カラム入力
DATA ステップにおける入力スタイルで、INPUT ステートメントにカラム番号を含めて、各変数の値がどのカラムに含まれているかを SAS に指示します。各変数の値がすべてのレコードで同じ場所にある場合、この入力スタイルが役立ちます。

カラムバイナリデータの記憶域
やや古いデータストレージ形式であり、現在ではあまり使用されていません。ほとんどの SAS System ユーザーにとっては必要ありません。カラムバイナリデータの記憶域では、データを圧縮することで、1 枚のパンチカードに 80 項目以上のデータを格納できます。複数パンチデッキおよびカードイメージのデータセットはまだ利用されることがあるため、SAS System には、カラムバイナリデータを読み込むための入力形式が用意されています。

監査証跡
ユーザー作成可能なオプション指定の SAS ファイル。SAS データファイルの変更情報をこのファイルに記録できます。オブザベーションが追加、削除、更新されるたびに、修正者、修正内容、修正日時に関する履歴情報が監査証跡ファイルに記録されます。

キー変数
プライマリファイルと参照ファイルの両方に存在し、SAS データセットにインデックスを付けるために使用可能な変数。

キーワード
SAS キーワードを参照。

キーワードパラメータ
名前で識別されるマクロパラメータの一種。名前の後に等号を付けます。複数のキーワードパラメータを任意の順序で指定できますが、位置パラメータの後に指定する必要があります。

ギガバイト
10 億バイトに相当するデジタル情報の測定単位。

記述統計量
推断するのではなく、値のコレクションからその特性を表す数量。記述統計量の種類は、中心化傾向、値の変動の指標、値分布の形状の指標です。

基数点
位取り表記法で、整数部の数字と小数部の数字を区切る記号。たとえば、10 進法では、基数点は小数点と呼ばれます。

キャリッジコントロール文字
用紙送りの行数や、改ページ、行のスキップ、重ね打ちのための現在行保持の時点をプリンタに指示する記号。

切り替え
処理機能のオンとオフの切り替えを可能にするオプション、パラメータまたはその他のメカニズム。

クエリ
1 つ以上のデータソースから特定の情報を要求する命令セット。

区切り文字
テキスト文字列の要素を区切る境界として機能する文字。

区切り文字を区別するデータ
個々のデータ値に引用符、カンマ、タブなどの埋め込み区切り文字が含まれているデータ。

クライアント
サーバーからのリソースかサービスのどちらかを(場合によってはネットワーク経由で)要求するアプリケーション。

クライアントセッション
クライアントコンピュータで実行する SAS セッション。クライアントセッションは、SAS ステートメントを受け取り、処理のためサブミットしてサーバーに渡します。クライアントセッションでは、クライアントセッションとサーバーセッション両方からの出力およびメッセージが管理されます。

グローバルスコープ
SAS マクロプログラミングで、グローバルマクロ変数を参照するためのコンテキスト境界が広範囲であることを示します。すなわち、現在の SAS セッションまたは SAS バッチプログラム内の任意の場所で参照できます。

グローバルマクロ変数
同じ名前のローカルマクロ変数が存在する場合を除いて、SAS プログラムでグローバルスコープかローカルスコープのどちらかで参照できるマクロ変数。グローバルマクロ変数は、当該セッションまたはプログラムの終了まで存在します。

クロス環境データアクセス
あるディレクトリベースの動作環境で作成された SAS データファイルを、別のディレクトリベースの環境で実行されている SAS セッションで読み取れるようにする SAS ソフトウェアの機能。

経過時間
コンピュータプログラム結果の受け取りに要する時計時間。これには、コンピュータによるプログラム実行に使用された時間や、プログラムがリソース(メモリやプリンタなど)の待機に費やした時間などが含まれます。

警告
SAS ログまたはメッセージウィンドウのメッセージ。プログラムの処理を続行するために SAS が修正アクションを行ったことを示します。

欠損値
特定の行または列にデータが含まれていない変数値の種類。デフォルトでは、数値変数の欠損値は 1 つのピリオドに、文字変数の欠損値はブランクに置き換えて書き込まれます。

更新
マスタデータセットの変数値をトランザクションデータセットのオブザベーションの値で置換するプロセス。

構成ファイル
SAS の実行環境を定義する SAS システムオプションを含む外部ファイル。これらのシステムオプションは、SAS 起動時に毎回有効になります。

構造化照会言語
標準化された高水準の照会言語。リレーショナルデータベース管理システムで使用され、データベース管理システムのオブジェクトを作成、操作します。SQL は SQL プロシジャによって実装されます。

後置アットマーク
入出力行を保持し、以降の INPUT または PUT ステートメントで SAS が読み取りや書き込みできるようにするため使用される特殊記号@。

構文エラー
SAS ステートメントのスペルまたは文法のエラー。構文エラーは、実行前の各 SAS ステップのコンパイル時に検出されます。

国際標準化機構(International Organization for Standardization)
規格の策定を推進し、世界各国間での製品、サービス、情報の共有を促進するための関連活動を後援する組織。

コマンド型マクロ
%MACRO ステートメントの CMD オプションで定義されるマクロ。

コメント
コメントステートメントを参照。

コメントステートメント
SAS プログラムに埋め込まれ、説明文として役に立つ情報。処理時にコメントは無視されますが、SAS ログに書き込まれます。コメント構文にはいくつか形式があります。たとえば、コメントは、アスタリスクで始まってセミコロンで終わるステートメント(例: *メッセージ;)として表示されることがあります。

コントローラ
コンピュータと周辺機器(ディスクや RAID など)間の相互作用を管理するコンピュータコンポーネント。たとえば、コントローラは CPU とディスクドライブ間のデータ入出力を管理します。1 つのコンピュータには、多数のコントローラが含まれています。単一 CPU から 2 つ以上のコントローラにコマンドを発行でき、単一コントローラから複数のディスクにコマンドを発行できます。

コンパイル
プログラムコンパイルを参照。

コンパイル済みストアド DATA ステッププログラム
コンパイル後 SAS ライブラリに格納された DATA ステッププログラムが含まれている SAS ファイル。コンパイル済みストアドプログラムは、すでにコンパイル済み

の中間コードを含んでいるので、必要になったときに再コンパイルせずに直接実行することができます。

サーバー
要求側クライアントにリソースかサービスのどちらかを(場合によってはネットワーク経由で)提供するソフトウェア。

サイト番号
SAS ソフトウェアのライセンス先の企業または組織の識別に使用される番号。サイト番号は、すべての SAS セッションでログの上部付近に表示されます。

サブクエリ
別のクエリ式の一部としてネストされたクエリ式。サブクエリが含まれる句によっては、そのサブクエリで 1 つまたは複数の値を戻すことができます。

算術演算子
SAS で、算術演算式において加算、減算、除算、乗算、累乗を実行するために使用するいずれかの記号(+、-、/、*、**)。SYSTEM 2000 ソフトウェアでのみ、**がサポートされません。

算術演算式
SAS 式を参照。

参照一貫性制約
あるデータファイルにある主キーの一貫性制約が、別のデータファイルにある外部キーの一貫性制約によって参照される場合に作成される一貫性制約。外部キーの一貫性制約は、外部キーデータファイル内にある 1 つ以上の変数のデータ値を、主キーデータファイルの対応する変数や値にリンクします。

シード
乱数関数または CALL ルーチンで乱数値計算に使用する初期値。

時間値
SAS 時間値を参照。

時間間隔
経過した時間を時、分または秒でカウントするための単位。

時間定数
SAS 時間定数を参照。

指数
数式で、基本数または基本式を何乗するかを示す数字または式。たとえば、.1234 * 10 の 4 乗という式では、指数は 4 になります。

システムオプション
SAS システムオプションを参照。

システム管理機能
z/OS および OS/390 オペレーティングシステムの機能。ジョブの実行時にオペレーティングシステムが使用するコンピューティングリソースについて情報を提供します。

実行時エラー
ステップのコンパイル時ではなくステップの実行時に発生するエラー。

実行ステートメント
DATA ステップにおいて、DATA ステップのコンパイル時ではなく、DATA ステップの実行時になんらかのアクションを起こさせる SAS ステートメント。

実行モード
これには、バッチモード、対話型モード(SAS ウィンドウ環境またはその他グラフィカルユーザーインターフェイスを使用)、対話型ラインモード、非対話型モードなどがあります。SAS での実行または SAS との対話の方法。

自動保存ファイル
プログラムエディタまたは拡張エディタの内容のコピー。SAS では、デフォルト時間間隔かまたはプリファレンスダイアログボックスで指定した時間間隔でエディタの内容が自動保存されます。

自動マクロ変数
ユーザーではなく SAS によって定義されるマクロ変数で、SAS セッションについての情報を提供します。たとえば、SYSPROCESSID 自動マクロ変数には、現在の SAS プロセスのプロセス ID が含まれます。

自動呼び出し機能
マクロを定義するソースステートメントを格納し、必要に応じてそのマクロを呼び出せる SAS の機能。プログラムに定義を含める必要はありません。

自動呼び出しマクロ
未コンパイルのソースコードおよびテキストが自動呼出しマクロライブラリに格納されているマクロ。自動呼び出しマクロは、コンパイル済みストアドマクロとは異なり、初めての呼び出し時の実行前にコンパイルされます。

修飾リスト入力
INPUT ステートメントで入力形式およびフォーマット修飾子と呼ばれる特殊命令を使用する入力スタイル。修飾リスト入力では、入力レコードをスキャンして、少なくとも 1 つのブランク(またはその他の区切り文字)か、場合によっては複数のブランクによって区切られたデータ値を探します。

集約記憶域
オペレーティングシステムで個別ファイルのグループを格納できる場所。この場所は、動作環境によって、ディレクトリ、フォルダ、区分データセットのように異なる名前で呼ばれます。

主キーの一貫性制約
汎用一貫性制約の一種。指定の変数には固有のデータ値が含まれていることが必要で、ヌルのデータ値は許可されません。データファイルには、主キーを 1 つのみ含められます。あるデータファイルの主キーの一貫性制約が、別ファイルにある外部キーの一貫性制約で参照される場合、その主キーの一貫性制約は参照一貫性制約です。

出力オブジェクト
DATA ステップまたは PROC ステップで生成されたデータが含まれるプログラミングオブジェクト。そのデータのフォーマット方法についての情報を提供するテーブル定義も含むことが可能です。

出力先
ODS 出力先を参照。

出力バッファ
DATA ステップにおいて、PUT ステートメントが指定したファイルや出力デバイスに書き出す前の書き出し先メモリ領域。

条件
SAS プログラムにおける 1 つ以上の数値式または文字式で、この結果値に基づいてなんらかの決定が下されます。

除外リスト
指定の ODS 出力先から除外する出力オブジェクトを ODS に示すリスト。

ジョブストリーム
指定された順序で実行される一続きの関連するプログラム。

シンボリック変数
マクロ変数を参照。

シンボルテーブル
マクロプロセッサが特定スコープのすべてのマクロ変数およびマクロステートメントラベルを格納する領域。

数値
通常は数字のみを含む値。指数表記および 16 進表記の数字が含まれます。数値には、小数点(.)、正符号(+)または負符号(-)が付く場合があります。数値は数値変数に格納されます。

数値出力形式
特定のパターンを使用して数値変数の値を書き出すよう SAS に指示する命令セット。

数値定数
SAS 式で使用される数字。

数値入力形式
特定のパターンを使用して数値データ値を読み取るよう SAS に指示する命令セット。

スタイル
ODS スタイルを参照。

ステートメント
SAS ステートメントを参照。

ステートメントオプション
特定の SAS ステートメントで指定する語であり、そのステートメントが実行する処理にのみ影響を与えます。

ステートメント型のマクロ
%MACRO ステートメントの STMT オプションで定義されるマクロ。

ステートメントラベル
後ろにコロンが付く SAS 名。DATA ステップであるステートメントの前に置くと、必要に応じて、別のステートメントがそのステートメントの実行を直接指示して、ステップ内の他のステートメントをスキップできます。

ステップ境界
SAS プログラムにおいて DATA ステップまたは PROC ステップの完了を SAS が認識する箇所。

スレッド
オペレーティングシステムでスケジュール可能な最小処理単位。

スレッド化
データ処理またはデータ I/O のいずれかに対するハイパフォーマンス技術。これにより、1 つのタスクが複数のスレッドに分割され、スレッドは 1 つ以上の CPU 上の複数のコアで同時に実行されます。

スレッド化 I/O
速度を上げるために複数スレッドで実行される I/O。スレッド化 I/O によってパフォーマンスを大幅に向上させるためには、I/O を実行しているアプリケーションにデータを迅速に処理する能力があることも必要です。

スレッド化処理
CPU バウンドアプリケーションの速度を改善するために複数スレッドで実行される処理。

スレッド対応プロシジャ
スレッド化 I/O またはスレッド化処理をサポートする SAS プロシジャ。

世代グループ
元のデータセットに対する置き換えの履歴を表すデータセットのグループ。世代グループは、ベースバージョンと一連の履歴バージョンで構成されます。

世代データセット
データセットの更新時に作成される SAS データセットのアーカイブバージョン。SAS データセットの複数のコピーが世代グループに格納されます。各データセットは同じルートメンバ名を持ちますが、バージョン番号はそれぞれ異なります。

世代番号
世代グループで、履歴バージョンの特定の 1 つの世代(バージョン)を示す数で、1 ずつ増加します。たとえば、AIR#272 という名前のデータセットの世代番号は 272 です。

セッション
ソフトウェアアプリケーションが使用されている一期間。アプリケーションの起動時から実行終了までを指します。

接頭演算子
変数、定数、関数、またはかっこ付きの式の直前に記述する演算子(-6*a における負符号など)。

宣言ステートメント
SAS に情報を提供するステートメント。プログラムステートメントのコンパイル時に有効になります。

ソートインジケータ
データファイルの属性。データセットが並べ替えられているかどうか、どのように並べ替えられたか、および並べ替えが検証されたかどうかを示します。具体的には、ソートインジケータ属性では次の情報が示されます。1)並べ替えで使用された BY 変数、2)文字変数で使用された文字セット、3)使用された文字変数の照合順序、4)並べ替え情報が検証されたかどうか。この属性は、データファイルディスクリプタ情

報に格納されます。処理の一環としてデータが並べ替え済みであることを要求する SAS プロシジャは、ソートインジケータを使用します。

相関
一方の変数値が増加したり減少したりすると、他方の変数値が大きくなったり小さくなったりする傾向がある 2 変数の関係。

相関係数
2 つの時系列値の線形関係の強さを測定する統計量。相関係数の値の範囲は-1 から 1 までです。

挿入演算子
オペランドの間に置く記号で、両側の 2 つのオペランドや片側の 1 つのオペランドに適用される演算を示します(8 > 6 の大なり記号や A + B の正符号など)。挿入演算子の一般的な種類は、算術、比較、論理(ブール)、その他(最小、最大および連結)の 4 つです。

双方向テキスト
アラビア語やヘブライ語などの書記体系のテキストは、一般的に右から左に記述されます。ただし、数字、および左から右に記述される他の言語で書かれた埋め込みテキストは例外です。

属性
変数の属性を参照。

ターゲット変数
関数または式の結果の割り当て先変数。

対称型マルチプロセッシング
I/O および処理の速度向上が見込まれるハードウェアおよびソフトウェアアーキテクチャ。SMP マシンには、複数の CPU とスレッド対応オペレーティングシステムがインストールされます。通常、SMP マシンは、複数のコントローラと、コントローラごとに複数のディスクドライブで構成されます。

タイトル
SAS 出力または SAS ログの各ページの上部に出力される見出し。

タイムスタンプ
特定のイベントが発生した日付と時間の記録。

ダイレクトアクセス
データセットをシーケンシャルに検索して一致するオブザベーションを見つけるのではなく、データセット内のデータレコードを直接取り出す手法。テーブルルックアップアプリケーションでは、オブザベーション番号によりレコードが取り出されます。

対話型ラインモード
SAS プログラムを SAS セッションプロンプトで 1 行ずつ入力する、SAS プログラム実行方法。Enter キーまたは Return キーを押すと即時に各行が処理されます。プロシジャ出力と情報メッセージはディスプレイデバイスに直接返されます。

タグセット
SAS 出力形式からマークアップ言語出力を作成する方法を定義するテンプレート。タグセットによって、ハイパーテキストマークアップ言語(HTML)、拡張可能マークアップ言語(XML)、LaTeX などのマークアップ出力が生成されます。

多次元データベース(MDDB)
あらかじめ要約しクロス集計したデータを、リレーショナルデータベーステーブルの行列形式ではなく、マトリックス形式で個別セルとして格納する特殊データストレージ構造。ソースデータはデータウェアハウスまたは他のデータソースから集められます。MDDB によって、ユーザーは大量の要約データにおける複数の関係を迅速に制限なく表示できます。

多次元配列
少なくとも 2 次元で、同じデータ型の変数を 1 つの名前にグループ化すること。処理時には、この変数グループ化によって、列、行、および(配列に応じて)高次元の結果がもたらされます。

単一インデックス
1 つの変数のみの値を使用してオブザベーションを検索するインデックス。

単純式
演算子を 1 つだけ使用する SAS 式。

中央演算処理装置
コンピュータの主要ハードウェアコンポーネント。CPU はプログラム命令を実行し、コンピュータのその他の部分の操作を制御します。

データ値
データレコードに 1 項目として格納される、文字、数値または英数字の情報の単位。

データエラー
SAS プログラムが無効な値を含むデータを分析したときに発生する実行エラーの一種。たとえば、文字データのための INPUT ステートメントで数値変数を指定した場合、データエラーが発生します。このエラーは SAS ログにレポートされますが、プログラムの実行は続行されます。

データ区分
データを含む物理ファイル。SAS Scalable Performance Data Engine データセットのデータコンポーネントを構成する物理ファイルのコレクションの一部です。

データ制御ブロック
z/OS などの IBM メインフレームオペレーティングシステムで、オペレーティングシステムデータセットの物理的特性についての情報が含まれる記憶領域。

データセット
SAS データセットを参照。

データセットオプション
SAS データセットオプションを参照。

データビュー
SAS データビューを参照。

データ表現
特定の動作環境におけるデータの保存形式。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 390 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)、ワードアラインメント(4 バイト境界か 8 バイト境界か)、データ型の長さ(16 ビット、32 ビット、64 ビットのいずれか)などは、動作環境ごとに決定されます。

データベース
編成された関連データのコレクション。データベースには通常、名前付きファイル、名前付きオブジェクト、またはテーブル、ビュー、インデックスなどのその他の名前付きエンティティが含まれます。

データベース管理システム
データベース形式で格納されているデータの作成および操作を可能にするソフトウェアアプリケーション。

テーブル定義
Output Delivery System (ODS)での出力のフォーマット方法を記述する命令セット。

テーブルルックアップ
主要ソースの変数値に基づいて、補助ソースから情報を取得する処理技術。

定数
SAS ソフトウェアにおいて、固定された値を表す数字または文字列。

定数テキスト
マクロの一部またはマクロ変数の値としてオープンコードで格納される文字列。これを使用して、マクロプロセッサは、SAS ステートメント、ディスプレイマネージャコマンド、または他のマクロプログラムステートメントとして使用されるテキストを生成します。

ディスク
ディスク上の任意のレコードに直接アクセスできるデータストレージのデバイス。ディスク記憶装置とは違い、テープ記憶装置ではレコードのシーケンシャル処理しか行えません。

ディスクリプタ情報
SAS データセットのコンテンツおよび属性についての情報。たとえば、ディスクリプタ情報には、変数のデータ型や長さ、データの作成に使用されたエンジンなどが含まれます。SAS は、ディスクリプタ情報の作成および管理をすべての SAS データセット内で行います。

デカルト積
結合テーブルごとの各行とそれ以外のすべての結合テーブルの各行とを適合させる結合の種類。

テキスト編集コマンド
特定のテキストエディタで使用されるコマンド。

テキスト文字列
文字列(character string)を参照。

デフォルトディレクトリ
ユーザーが作業を行っているディレクトリ。ログイン時のデフォルトディレクトリは通常、ホームディレクトリになります。

テンプレートストア
TEMPLATE プロシジャで作成された定義を含むアイテムストア。SAS 提供の定義は、アイテムストア Sashelp.Tmplmst 内にあります。ユーザーが作成した定義は、書き込み権限のある任意のテンプレートストアに格納できます。

トークン
SAS が入力処理できるように、SAS 言語またはマクロ言語が入力を分割する際の単位。トークン(ワードとも呼ばれる)には、英単語に似た項目(変数名など)および、似ていない項目(算術演算子やセミコロンなど)が含まれます。

動作環境
コンピュータまたはパーティションに対して使用可能な、コンピュータ、またはコンピュータの論理パーティション、ならびにリソース(オペレーティンングシステムならびにその他のソフトウェアおよびハードウェアなど)。

トランザクションデータセット
更新操作において、マスタデータセットを更新するために必要とされる情報を含むデータセット。

トランスコード
SAS ファイルのコンテンツを、あるエンコーディングから別のエンコーディングに変換する処理。セッションエンコーディングとファイルエンコーディングが異なる場合(UNIX の Latin 1 エンコーディングから IBM メインフレームのドイツ語 EBCDIC エンコーディングにデータを転送する際など)、トランスコードが必要になります。

トレードオフ
一方のリソースの使用を削減するために、プログラムが他方のリソースの使用を増やせること。

名前付き出力
PUT ステートメントで等号を使用し、variable=data-value の形式で変数値を書き出すスタイル。

名前付き入力
INPUT ステートメントで等号を使用し、variable=data-value の形式でデータ値を読み取るスタイル。

名前リテラル
引用符内の文字列と、その後の大文字または小文字 n で表現される名前トークン。名前リテラルを用いると、SAS データセットまたは変数を指定する際、本来は SAS 名に使用できない特殊文字(ブランクなど)を使用できるようになります。閉じ引用符と n との間のブランクは、名前リテラルを指定する場合は無効です。VALIDVARNAME=V7 で使用できない文字がデータセットまたは変数の名前リテラルに含まれている場合は、VALIDVARNAME=システムオプションを ANY に設定する必要があります。システムオプションを ANY に設定しても、V6 Engine は、ブランクが間に入っている名前はサポートしません。

二重後置アットマーク
DATA ステップの反復を複数回実行する間に、一行のデータを入力バッファに保持するために使用する特殊記号@@。

日時値
SAS 日時値を参照。

日時間隔
経過した時間を時、分、秒、日、月でカウントするための単位。

日時定数
SAS 日時定数を参照。

ニブル
半バイトすなわち 4 ビット(2 進数)。

入出力時間
処理対象データを記憶域からメモリに移動する処理、およびメモリからの結果を記憶域、ディスプレイまたはプリンタに移動する処理で費やされた時間。

入出力操作
ディスクやテープなどの記憶媒体から物理的にデータを読み取ったり、または記憶媒体にデータを書き込んだりする、すべての操作。

認証プロバイダ
ユーザーの識別および認証に使用されるソフトウェアコンポーネント。たとえば、LDAP サーバーまたはホストオペレーティングシステムが、認証を提供できます。

ヌルステートメント
1 つのセミコロンかまたは 4 つのセミコロンから成るステートメント。ヌルステートメントは、DATA ステップで入力ストリームデータの最後を指定するために最もよく使用されます。

ネイティブビュー
DATA ステップまたは SQL プロシジャを使用して作成する SAS データビュー。

ネイティブライブラリエンジン
SAS によってのみ作成と処理が行われる種類の SAS ファイルにアクセスする SAS エンジン。

ハイパーテキストマークアップ言語
テキストファイルでのテキストのレイアウトやスタイルをコードで示すコーディングシステム。その他の HTML コードでは、画像、音、ビデオストリーム、アプレット(小規模なソフトウェアアプリケーション)などの電子オブジェクトを HTML ドキュメントに埋め込むことができます。HTML ドキュメントはすべての Web ブラウザで処理できます。

配列参照
配列で処理される要素の参照。

配列名
変数または一時データ要素のグループを識別するために選択された名前。同じ DATA ステップまたは SCL (SAS コンポーネント言語)プログラム内の変数名とは異なる、有効な SAS 名にする必要があります。

パック 10 進データ
各バイトが 2 桁の 10 進数を表す 10 進数のエンコーディング方法。

バッチモード
SAS プログラムの非対話型実行方式。これにより、ファイル(必要なオペレーティングシステムコマンドすべてに加えて SAS ステートメントを含む)が、実行のために動作環境のバッチキューにサブミットされます。

パフォーマンス統計量
コンパイルおよび実行時にプログラムで使用されるリソースの統計量を含むデータ。

汎用一貫性制約
1 つのファイル内の変数値を制限できる一貫性制約。汎用一貫性制約には、check、not null、unique、primary key の 4 種類があります。

ピクチャ
FORMAT プロシジャで、数値変数の値を出力するためのテンプレート。

非対話型処理
非対話型モードを参照。

非対話型モード
ユーザーが SAS ステートメントのファイルを準備して、オペレーティングシステムにプログラムをサブミットする、SAS プログラムの実行方法。プログラムはすぐに実行され、現在のセッションを構成します。

ビッグエンディアン
コンピュータメモリ上で、重要性の高いバイトが重要性の低いバイトよりも小さいストレージアドレスに書き込まれるバイト順序。SAS System では、IBM メインフレーム、HP-UX、AIX、Solaris の各プラットフォームがビッグエンディアンと見なされます。

ビュー
後で使用するために名前を付けて格納される仮想データセットの定義。ビューにデータは含まれません。他の場所に格納されたデータが単に記述または定義されるだけです。

ビューディスクリプタ
アクセスディスクリプタによって記述された DBMS データの一部または全部を定義する SAS/ACCESS ファイル。

標準化
特定の平均値と標準偏差に基づいて変数の値を別の尺度に変換する方法。

標準データ
数字 1 桁または 1 文字が 1 バイトのストレージを占有するデータ。

ファイルアクセス方式
ファイルからの読み取りやファイルへの書き出しにエンジンが使用する命令セット。

ファイル参照名(file reference)
ファイル参照名(fileref)を参照。

ファイル参照名(fileref)
外部ファイルか、またはディレクトリもしくはフォルダなどの集約記憶域に一時的に割り当てられる名前。SAS システムでは、ファイル参照名を使用してファイルや保存場所を識別します。

ファイルショートカット
ファイル参照名を表す Microsoft Windows 用語。

ファイル転送プロトコル
ネットワークを介したコンピュータ間でのファイル転送に使用される通信プロトコル。

ブール演算子
評価時にブール値を生成するプログラミング言語内の式。

ブール数式
ブール演算子を使用して真か偽のいずれかの条件を表すステートメント。真の値は 1 で、偽の値は 0 です。

ファンクションキー
特定のソフトウェア環境における特定のアクションを定義できるキーボードキー。

フォーマッティング文字
CALENDAR、FREQ、TABULATE などの各種プロシジャで定義されている、表形式出力用の垂直線や水平線線の作成に使用される+、|、-などの文字。

フォーマット修飾子
INPUT ステートメントと PUT ステートメントで使用される特殊記号。これにより、入力データの読み取りと出力データの書き出しを制御できます。

フォーマット出力
PUT ステートメントで SAS 出力形式を使用して、変数値を書き出す方法を指定する出力スタイル。

フォーマット入力
INPUT ステートメントで入力形式と呼ばれる特殊命令を使用して、データフィールドに入力された値の解釈方法を決定する入力スタイル。

フォームレイアウト
連続用紙のページ(フォームページ)での宛名ラベルなどのフォーム単位の数と配置。

複合 WHERE 式
演算子が 2 つ以上含まれる WHERE 式(例: WHERE X=1 and Y>3)。

複合インデックス
2 つ以上のキー変数の値を検証することによって、SAS データセットのオブザベーションの場所を特定するインデックス。

複合式
演算子が 2 つ以上含まれる式。

物理的な順序
データレコードまたはオブザベーションの格納構造における順序。

物理ファイル名
オペレーティングシステムがファイルを識別するために使用する名前。

浮動小数点表現
指数表記と同様、コンピュータ上で実数を格納するコンパクト形式。異なるオペレーティングシステムでは、浮動小数点表現について異なる手法が使用されます。

プラットフォーム
プログラムが実行される動作環境(オペレーティングシステムとコンピュータハードウェアの両方を含む)。

プログラムコンパイル
構文をチェックし、プログラム部分をコンピュータで実行可能な形式に変換する処理。

プログラムデータベクトル
コンピュータメモリの一時領域。ここに SAS データセットのオブザベーションが 1 つずつ作成されます。プログラムデータベクトルは論理的概念であり、必ずしも一続きのメモリ領域に相当するわけではありません。

プロシジャ
SAS プロシジャを参照。

プロシジャ出力ファイル
SAS プロシジャで実行された分析結果、またはプロシジャで生成されたレポートが含まれる外部ファイル。ほとんどの SAS プロシジャでは、デフォルトでプロシジャ出力ファイルに出力が書き出されます。SAS DATA ステップで PUT ステートメントおよび FILE ステートメントを PRINT 出力先と一緒に使用して生成されたレポートも、このファイルに送られます。

ブロック
データセット内のオブザベーションのグループ。ブロックを使用すると、スレッド対応アプリケーションでは、個別オブザベーションとして配信された場合よりも高速でオブザベーションの読み取り、書き込みおよび処理ができます。

プロファイルカタログ
Sasuser.Profile カタログを参照。

分位点
データを分割して同数のオブザベーションを含むグループにする任意の点または値、あるいはそのグループ。四分位点、五分位点、百分位点はすべて分位点の例です。

分類変数(class variable)
分類変数(classification variable)を参照。

分類変数(classification variable)
この変数の値を使用して、データセット内のオブザベーションを、分析に有効な異なるグループに分類します。分類変数には、文字値か数値のどちらかを設定できます。分類変数には、グループ、サブグループ、カテゴリ、BY 変数が含まれます。

ページサイズ
1 回の入出力操作で外部ストレージとメモリ間を SAS が移動させるデータのバイト数。ページサイズは、SAS データセットのバッファサイズと似ています。

ページ次元
TABULATE プロシジャにおいて、テーブル内のページ数や配置を決定する変数、変数値、および統計量の組み合わせ。

ベースバージョン
最後に作成された最も新しいバージョンの世代データセット。ベースバージョンの名前には、世代番号を示す 4 文字の接尾辞は含まれません。

並列 I/O
複数の CPU と複数のコントローラを利用する入出力方法。コントローラごとに複数のディスクを使用して独立したスレッドでデータを読み書きします。

並列処理
大きなジョブを複数の小さなジョブに分割して、複数の CPU で並列実行できるようにする処理方法。

ヘッダー
MIME メッセージや HTTP メッセージなどのインターネットプロトコルメッセージの一部。メッセージについての情報を提供します。ヘッダーには、件名、返信先アドレス、SMTP 送信元アドレス、差出人アドレスを含められます。

ヘッダールーチン
印刷ファイルのページヘッダーを生成する DATA ステップステートメントのグループ。ヘッダールーチンは、ステートメントラベルで始まり、RETURN ステートメントで終わります。FILE ステートメントの HEADER=オプションとは同じとみなせます。

変数
SAS 変数を参照。

変数属性
名前、ラベル、出力形式、入力形式、データ型、長さなど、特定の変数に関連付けられた特性。

変数の種類
数値か文字のどちらかの変数分類。種類は SAS 変数の属性です。

変数ラベル
256 文字までの説明テキスト。変数名のかわりか、または変数名に追加して、特定のプロシジャで出力に表示できます。

ポインタ
DATA ステップで、入出力バッファ内の位置を追跡するために SAS が使用するプログラミングツール。

ポインタコントロール
データを読み書きする前にポインタの移動を SAS に指示する処理。

ホスト
ホスト動作環境を参照。

ホスト動作環境
IP アドレスまたはドメイン名によって識別され、ソフトウェアアプリケーションに対する集中制御機能を提供する動作環境(コンピュータ、オペレーティングシステムおよびそれ以外のソフトウェアとハードウェア)。

マークアップ言語
レイアウトや特定のコンテンツを定義するために、テキストに埋め込まれたコードセット。

マージ
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 SAS データセットの単一のオブザベーションにする処理。

マクロ
コンパイル済みプログラムステートメントと保存されたテキストのグループを含む SAS カタログエントリ。

マクロ関数
マクロ機能によって定義される関数。各マクロ関数で、1 つ以上の引数が処理され、結果が出されます。

マクロ起動
マクロ呼び出しを参照。

マクロ機能
SAS プログラムの拡張やカスタマイズに使用できる Base SAS ソフトウェアのコンポーネント。マクロ機能を使用すると、一般的なタスクを実行するのに入力する必要のあるテキスト量を削減できます。マクロ機能は、マクロプロセッサとマクロプログラミング言語で構成されています。

マクロ言語
マクロプロセッサとの通信に使用されるプログラミング言語。

マクロ式
実行時に値を返す任意の有効な記号組み合わせ。テキスト、論理および算術の 3 種類のマクロ式があります。テキスト式は、解決(実行)時にテキストを生成し、テキスト、マクロ変数、マクロ関数、マクロ呼び出しの任意の組み合わせから成ります。論理式は、論理演算子とオペランドから成り、真か偽どちらかの値を返します。算術式は、算術演算子とオペランドから成り、数値を返します。

マクロの実行
コンパイル済みのマクロプログラムステートメントにより与えられる命令に従って、テキストの生成、SAS ログへのメッセージの書き込み、入力の受け入れ、マクロ変数値の作成や変更、その他のアクティビティの実行などを行うこと。生成されるテキストには、SAS ステートメント、SAS コマンド、その他のマクロプログラムステートメントなどがあります。

マクロパラメータ
%MACRO ステートメントでかっこ内に定義されるローカルマクロ変数。マクロの起動時に、マクロパラメータに値を提供します。

マクロプロセッサ
マクロとマクロプログラムステートメントのコンパイルと実行を行う、SAS のコンポーネント。

マクロ変数
SAS マクロプログラミング言語に含まれる変数。マクロ変数の値は、ユーザーが変更しない限り一定のままの文字列です。

マクロ変数参照
マクロ変数の名前で、アンパサンドが前に付きます。マクロプロセッサは、指定したマクロ変数の値でマクロ変数参照を置換します。

マクロ呼び出し
コンパイル済みストアドマクロプログラムを起動するステートメント。

マスタデータセット
更新操作において、更新対象の情報を含むデータセット。

マッチマージ
BY 変数の値に従って 2 つ以上の SAS データセットのオブザベーションを結合するプロセス。

メタデータ
データベースに格納、管理されているデータについての説明データ。今後の使用のために、取得/アーカイブされたデータへのアクセスを容易にするためのものです。

メタデータオブジェクト
テーブル、サーバー、ユーザー、ネットワーク上の別リソースなどについて説明する属性セット。メタデータオブジェクトに含まれる固有の属性は、使用するメタデータモデルによって異なります。

メタデータサーバー
1 つ以上のクライアントアプリケーションにメタデータ管理サービスを提供するサーバー。

メニュー
ユーザーに選択肢を提示するウィンドウオブジェクト。SAS では、メニューには、メニューバー、プルダウンメニュー、ブロックメニュー、選択リストが含まれます。

メモリ
中央演算処理装置(CPU)がプログラムの演算に当てる作業領域のサイズ。

メンバタイプ
SAS ファイルに格納された情報の種類を識別する SAS 名。メンバタイプには、ACCESS、AUDIT、DMBD、DATA、CATALOG、FDB、INDEX、ITEMSTOR、MDDB、PROGRAM、UTILITY、VIEW があります。

文字値
アルファベット、0 から 9 までの数字、およびその他の特殊文字を含む値。

文字エンコーディング
言語または言語グループで使用される文字の集合。文字エンコーディングには、各国固有の文字、特殊文字、0 から 9 までの数字、制御文字が含まれます。

文字関数
関数の一種。文字列の操作、比較、評価、分析のいずれかが行えます。

文字出力形式
特定のパターンを使用して文字データ値を書き出すよう SAS に指示する命令セット。

文字セット
文字エンコーディング(character encoding)を参照。

文字定数
SAS ステートメントにおいて引用符で囲まれた文字列で、変数の名前ではなく固定値を示します。許容される最大文字数は 32,767 です。文字定数は文字リテラルと呼ばれることもあります。

文字入力形式
特定のパターンを使用して文字データ値を文字変数に読み込むよう SAS に指示する命令セット。

文字の比較
文字オペランドが左から右へ 1 文字ずつ比較されるプロセス。結果は数値になります。文字オペランドが等しい場合は結果が値 1 になり、等しくない場合は結果が値 0 になります。

文字変数
値がアルファベットおよび特殊文字ならびに数字から成る変数。

文字リテラル
文字定数(character constant)を参照。

文字列(character string)
1 つ以上の連続した英数字またはその他キーボード文字、あるいはその両方。

文字列(string)
文字列(character string)を参照。

最も古いバージョン
世代グループで最も古い履歴バージョンのデータセット。

最も若いバージョン
世代グループでベースバージョンに時間的に最も近い履歴バージョンのデータセット。

モデルテキスト
定数テキストを参照。

ユニバーサル印刷
SAS 出力を、直接プリンタに送るだけでなく、PDF、Postscript、GIF、PNG、SVG、PCL ファイルにも送れる SAS の機能。ユニバーサル印刷システムは、出力のカスタマイズを可能にする多くのオプションも提供し、SAS がサポートするすべての動作環境で使用可能です。

要約統計量
記述統計量を参照。

ライブラリエンジン
ファイルのグループにアクセスし、SAS Utility のウィンドウや SAS プロシジャで処理するため適正な形式にそれらを変換して配置するエンジン。ライブラリエンジンでは、ライブラリの基本的な処理の設定や、ライブラリディレクトリのファイルリストの表示も行います。

ライブラリ参照名(library reference)
ライブラリ参照名(libref)を参照。

ライブラリ参照名(libref)
SAS ライブラリの場所に関連付けられる SAS 名。たとえば、MYLIB.MYFILE という名前では、MYLIB がライブラリ参照名で、MYFILE が SAS ライブラリ内のファイルです。

ラインホールド指定子
INPUT ステートメントと PUT ステートメントで使用される特殊記号。これにより、以降の処理のために入力バッファまたは出力バッファにレコードを保持できます。ラインホールド指定子には、後置アットマーク(@)および二重後置アットマーク(@@)が含まれます。

ラインモード
対話型のラインモードを参照。

ランダムアクセス
SAS データモデルのアクセスパターン。これにより、オブザベーションは、すべてが順番に処理されるのではなく、あるインジケータ変数の値に従って処理されます。

リスト入力
INPUT ステートメントでカラム位置ではなく変数名を指定する入力スタイル。リスト入力では、入力レコードをスキャンして、少なくとも 1 つのブランクかまたはその他の区切り文字で区切られたデータ値を探します。

リソース測定機能
z/OS および OS/390 オペレーティングシステムの機能。処理される各ジョブについて情報が記録されます。

リターン値
関数の実行結果を示す値。

リターンコード
要求が成功したかどうかを示す数値。リターンコードで、特定のエラーまたは警告を示すことができます。

リテラル
固定値を示す数字または文字列。

リトルエンディアン
コンピュータメモリ上で、重要性の低いバイトが重要性の高いバイトよりも小さいストレージアドレスに書き込まれるバイト順序。SAS System では、OpenVMS Alpha、Digital UNIX、Intel ABI、Windows の各プラットフォームがリトルエンディアンと見なされます。

履歴バージョン
以前のベースバージョンの履歴(古いコピー)。履歴バージョンの名前には、#003 など、世代番号を表す 4 文字の接頭辞が付いています。

ローカライズ
ソフトウェアを特定の地理文化地域(ロケール)に適合させる処理。ユーザーインタフェイス、システムメッセージおよびドキュメントの翻訳がローカライズ処理の大部分を占めます。

ログ
SAS ログを参照。

ロケール
地理的な地域の言語や地域規則および文化を反映させた設定。地域規則には、国または地域の、用紙サイズ、日付、時間、数字の出力形式や、通貨記号が含まれます。ロケール値の例としては、French_Canada、Portuguese_Brazil、Chinese_Singapore などが挙げられます。

論理演算子
複数の比較演算を結び付けるために式で使用される演算子。論理演算子は AND、OR および NOT です。

論理レコード長
外部ファイルの 1 行や SAS オブザベーションなどの関連データから成る情報の単位のバイト数。論理レコード長のデフォルト値は、動作環境によって異なります。

ワード
SAS プログラミング言語で、SAS に指示を伝達する文字の集合。SAS プログラムにおける最小単位です。SAS ワードは最大で 32,767 文字を格納できます。

貼り付けバッファ
STORE コマンドまたは CUT コマンドで格納されたテキストを保持する一時格納場所。貼り付けバッファの内容は、現在の SAS セッションでのみ有効です。

キーワード

T

U

か

さ

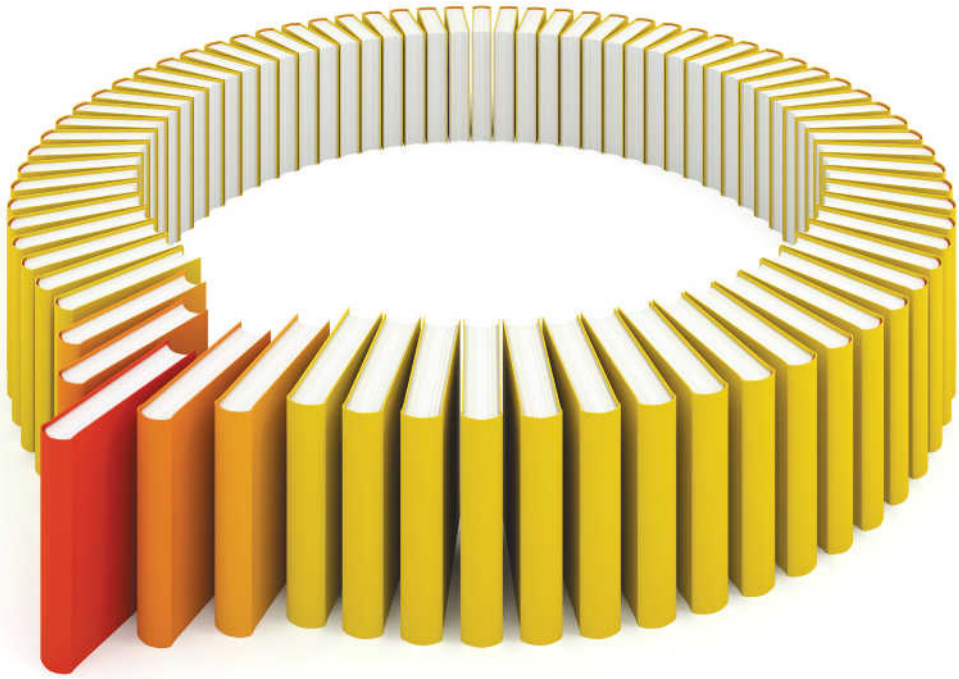
た

は

ら

わ

2



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

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

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