Natural for Windows

ステートメント

バージョン 6.3.3

October 2008

Natural

This document applies to Natural バージョン 6.3.3 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

目次

1 ステートメント

このドキュメントでは、Naturalプログラミング言語のステートメントについて説明します。次の項目で構成されています。

構文記号およびオペランド定義テーブル	ステートメントの構文を示す図で使用している記号、およびオペランド定義テーブルに関する情報を示します。
ステートメント使用関連トピック	『*プログラミングガイド*』の特別なトピックのリストを示します。
機能別ステートメント	機能別に分類されたNaturalステートメントの概要を示します。
ステートメント（アルファベット順）	ステートメント（SQLステートメントを除く）をアルファベット順で説明します。
Natural SQL ステートメント	SQLデータベース内のデータを管理するためにNaturalプログラムで使用できる特定のステートメントについて説明します。
参照プログラム例	Naturalステートメントおよびシステム変数のリファレンスドキュメントで参照されている追加のプログラム例を示します。 **注意:** 一般に、ステートメントの説明で示されているプログラム例はストラクチャードモードで記述されています。レポーティングモードとストラクチャードモードで構文が大幅に異なるステートメントについては、同等のレポーティングモードの例も参照として提供されています。これらのプログラム例は、Naturalライブラリ `SYSEXSYN` にソースコード形式で用意されています。Naturalステートメントのその他のプログラム例については、『*プログラミングガイド*』の「*参照プログラム例*」を参照してください。これらのプログラム例は、Naturalライブラリ `SYSEXPG` に用意されています。これらのライブラリが使用可能であるかをNatural管理者に確認してください。プログラム例では、Software AG提供のデモ用ファイル `EMPLOYEES` と `VEHICLES` のデータを使用します。

2 構文記号およびオペランド定義テーブル

このchapterでは、次のトピックについて説明します。

構文記号

Natural ステートメントの構文を示す図には次の記号を使用しています。

構文記号	説明
ABCDEF	大文字の用語は Natural キーワードまたは Natural 予約語のいずれかで、示されたとおりに入力する必要があります。
ABCDEF	大文字のオプション用語全体に下線が付いている（ハイパーリンクではない）場合は、その用語がデフォルト値であることを示します。 用語を省略すると、下線の付いた値が適用されます。
ABCDEF	大文字の用語の一部に下線が付いている（ハイパーリンクではない）場合は、下線部分がその用語の入力可能な省略形です。
abcdef	斜体の文字は、変数情報を表します。 この用語を指定する場合は、有効な値を入力する必要があります。 **注意:** *statement* または *statements* の代わりに、状況に応じて 1 つまたは複数のステートメントを指定する必要があります。不要なステートメントがある場合は、IGNORE ステートメントを挿入します。
[ ]	角カッコ内の要素はオプションです。 角カッコに複数の行が入れ子になって含まれている場合は、各行がオプションの選択肢です。 選択肢の中から 1 つ選択できます。
{ }	中カッコに複数の行が入れ子になって含まれている場合は、各行が選択肢です。 選択肢の中から 1 つのみ選択する必要があります。
|	選択肢は縦棒で区切られます。
...	省略記号の前の用語は任意に繰り返すことができます。 省略記号の後の数値は用語の繰り返し回数を示します。 省略記号の前の用語が角カッコまたは中カッコに囲まれている場合、省略記号はカッコ全体に適用します。
,...	コンマと省略記号の前の用語は任意に繰り返すことができます。繰り返す場合は、繰り返しをコンマで区切る必要があります。 コンマと省略記号の後の数値は用語の繰り返し回数を示します。 コンマと省略記号の前の用語が角カッコまたは中カッコに囲まれている場合、コンマと省略記号はカッコ全体に適用します。

構文記号	説明
:...	コロンと省略記号の前の用語は任意に繰り返すことができます。繰り返す場合は、繰り返しをコロンで区切る必要があります。 コロンと省略記号の後の数値は用語の繰り返し回数を示します。 コロンと省略記号の前の用語が角カッコまたは中カッコに囲まれている場合、コロンと省略記号はカッコ全体に適用します。
他の記号 （[ ] { } | ... ,... :... を除く）	他のすべての記号は、この表で定義しているものを除いて、示されたとおりに入力する必要があります。 *例外*：SQL スカラー連結演算子は 2 つの縦線で表現され、構文定義に示されたとおりに完全に入力する必要があります。

例：

- WRITE、USING、MAP、および FORM は Natural キーワードであり、示されたとおりに入力する必要があります。
- *operand1* および *operand2* はユーザーが指定する変数であり、処理するオブジェクトの名前を指定します。
- 中カッコは、FORM または MAP のうちどちらか 1 つを必ず選択する必要があることを示しています。
- 角カッコは、USING および *operand2* がオプションの要素であり、必要に応じて指定できることを示しています。
- 省略記号は、*operand2* を複数回指定できることを示しています。

オペランド定義テーブル

Natural ステートメントの構文に 1 つ以上のオペランドが含まれる場合、次の形式の表が示されています。

オペランド	構文要素						フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S	A	G	N/M	E	A	U	N	P	I	F	B	D	T	L	C	G	O	可／不可	可／不可

表内の意味は次のとおりです。

構文要素

オペランドで使用できる要素を示します。

C	定数
S	単一オカレンス（スカラ、つまり配列範囲でもグループでもないフィールド／変数）
A	配列
G	グループ
N/M	Natural システム変数 N = 全システム変数を使用できることを示します。 M = *変更可能*なシステム変数のみを使用できることを示します。 システム変数の内容が変更可能かどうかについては、Natural の『システム変数』ドキュメントを参照してください。
E	算術演算式

フォーマット

オペランドで使用できるフォーマットを示します。

A	英数字（ASCII コードページ）
U	英数字（Unicode）
N	アンパック型数値
P	パック型数値
I	整数
F	浮動小数点
B	バイナリ
D	日付
T	時刻
L	論理
C	属性制御
G	GUI のハンドル
O	HANDLE OF OBJECT

ステートメント参照

ステートメントラベルまたはソースコード行番号で参照できるかどうかを示します。

ダイナミック定義

プログラム内でフィールドをダイナミックに定義できるかどうかを示します。 レポーティングモードでのみ可能です。

3 ステートメント使用関連トピック

Natural ステートメントの使用方法の詳細については、Natural の『*プログラミングガイド*』で次の情報を参照してください。

- *ユーザー定義変数*
- *X-array*
- ダイナミック変数／フィールドとラージ変数／フィールド
  - *はじめに*
  - *使用方法*
- *ユーザー定義定数*
- *レポート指定*
- *テキスト表記*
- *ユーザーコメント*
- *論理条件基準*
- *演算割り当てのルール*
- *ユーザー定義関数*

4 機能別ステートメント

このchapterでは、機能別に分類されたステートメントの概要を示します。

このchapterでは、次のトピックについて説明します。

Notes:

1. ストラクチャードモードおよびレポーティングモードの両方で使用できるステートメントと、レポーティングモードでしか使用できないステートメントが存在します。『*プログラミングガイド*』の「*Natural プログラミングモード*」を参照してください。
2. DLOGOFF、DLOGON、SHOW、IMPORT、および EXPORT ステートメントは、Entire DB がインストールされている場合にのみ使用できます。詳細については、『*Entire DB*』ドキュメントを参照してください。

データベースへのアクセスと更新

Natural DML ステートメント

データベース内の情報にアクセスして操作するには、次の Natural データ操作言語（DML）ステートメントを使用します。

READ	データベースファイル内の物理的または論理的な一連のレコードを読み取ります。
FIND	ユーザー指定の条件に基づいて、データベースファイルからレコードを選択します。
HISTOGRAM	データベースフィールドの値を読み取ります。
GET	ISN（内部シーケンス番号）またはRNO（レコード番号）を指定してレコードを読み取ります。
GET SAME	現在処理中のレコードを再読み取りします。
ACCEPT/REJECT	ユーザー指定の条件に基づいて、レコードの受け入れ／除外を行います。
PASSW	パスワード保護されたファイルにアクセスするためのパスワードを指定します。
LIMIT	READ、FIND、または HISTOGRAM 処理ループの実行回数を制限します。
STORE	新しいレコードをデータベースに追加します。
UPDATE	データベース内のレコードを更新します。
DELETE	データベースからレコードを削除します。
END TRANSACTION	論理トランザクションの終了を示します。
BACKOUT TRANSACTION	完了していない論理トランザクションをバックアウトします。
GET TRANSACTION DATA	直前の END TRANSACTION ステートメントで保存したトランザクションデータを読み取ります。

RETRY	他のユーザー用にホールド状態になっているレコードの再読み取りを試みます。
AT START OF DATA	処理ループで一連のレコードの先頭が処理されるときに実行されるステートメントを指定します。
AT END OF DATA	処理ループで一連のレコードの最後が処理されたときに実行されるステートメントを指定します。
AT BREAK	制御フィールドの値が変更されるときに実行されるステートメントを指定します（ブレイク処理）。
BEFORE BREAK PROCESSING	ブレイク処理の実行前に実行されるステートメントを指定します。
PERFORM BREAK PROCESSING	直ちにブレイク処理を呼び出します。

Natural SQL ステートメント

Natural DML ステートメントの他に、Natural プログラムで SQL ステートメントを使用して直接 SQL を扱うことができます。

次の SQL ステートメントが有効です。

CALLDBPROC	NaturalからアクセスできるSQLデータベースシステムのストアドプロシージャを呼び出します。
COMMIT	論理トランザクションの終了を示し、トランザクション中にロックされた全データを解放します。 すべてのデータ変更がコミットされて確定されます。
DELETE	カーソルを使用しないでテーブル内の行を削除したり（「検索済」 DELETE）、カーソルが位置づけられたテーブル内の行を削除したり（「位置決め」 DELETE）します。
INSERT	1 つ以上の行をテーブルに新しく追加します。
PROCESS SQL	基準データベースに対して SQL ステートメントを発行します。
READ RESULT SET	前の CALLDBPROC ステートメントで呼び出されたストアドプロシージャによって作成された結果セットを読み込みます。
ROLLBACK	最後のリカバリ単位の開始以降に行われたデータベース更新をすべて元に戻します。
SELECT	任意の行数を取得するためのカーソル選択と、多くても1つの行を取得する非カーソル選択（単独 SELECT）の両方をサポートします。
UPDATE	カーソルを使用しないでテーブル内の行に対して更新処理を実行したり（「検索済」 UPDATE）、カーソルが位置づけられた行の列に対して更新処理を実行したり（「位置決め」 UPDATE）します。

算術演算とデータ移動操作

算術演算とデータ移動操作には次のステートメントを使用します。

COMPUTE	算術演算を実行するか、値をフィールドに割り当てます。
ADD	複数のオペランドを加算します。
SUBTRACT	1 オペランドから他の 1 つ以上のオペランドを減算します。
MULTIPLY	複数のオペランドを乗算します。
DIVIDE	1 オペランドを他のオペランドに除算します。
EXAMINE TRANSLATE	フィールド内の文字を大文字／小文字または他の文字に変換します。
MOVE	1 オペランドの値を 1 つ以上のフィールドに移動します。
MOVE ALL	値の複数オカレンスを他のフィールドに移動します。
COMPRESS	複数のフィールドの値を 1 フィールドに連結します。
SEPARATE	1 フィールドの内容を複数のフィールドに分割します。
EXAMINE	特定の値でフィールドをスキャンし、それを置き換えたり、出現回数をカウントしたりします。
RESET	フィールドの値を0（数値の場合）、空白（英数字の場合）、またはその初期値に設定します。

ループ実行

処理ループの実行には次のステートメントが関係します。

ESCAPE	処理ループの実行を停止します。
FOR	処理ループを開始し、ループの処理回数を制御します。
REPEAT	処理ループを開始し、指定された条件に基づいて終了します。
SORT	レコードをソートします。

出力レポートの作成

出力レポートの作成には次のステートメントを使用します。

FORMAT	出力パラメータ設定を指定します。
DISPLAY	列形式で出力するフィールドを指定します。
WRITE / PRINT	列形式以外で出力するフィールドを指定します。
WRITE TITLE	レポートの各ページの上に出力するテキストを指定します。
WRITE TRAILER	レポートの各ページの下に出力するテキストを指定します。
AT TOP OF PAGE	新しい出力ページの開始時に実行される処理を指定します。
AT END OF PAGE	出力ページの最後に達したときに実行される処理を指定します。
SKIP	レポートに 1 行以上の空白行を作成します。
EJECT	タイトルやヘッダーを含まずにページ送りを行います。
NEWPAGE	タイトルやヘッダー付きでページ送りを行います。
SUSPEND IDENTICAL SUPPRESS	1 レコードに対して同一値の省略を中止します。
DEFINE PRINTER	論理出力先にレポートを割り当てます。
CLOSE PRINTER	プリンタをクローズします。

対話型処理用の画面生成

データの対話型処理のためにデータ画面（マップ）を生成するには、次のステートメントを使用します。

INPUT	データを表示したり、入力したりするための形式化された画面（マップ）を作成します。
REINPUT	INPUT ステートメントを再実行します（前の INPUT ステートメントに対する入力データが誤りであった場合）。
DEFINE WINDOW	ウィンドウのサイズ、位置、および属性を指定します。
SET WINDOW	ウィンドウを稼動したり解除したりします。
PROCESS PAGE	Web リッチ GUI 画面に対するデータマッピングを作成します。
PROCESS PAGE USING	ページレイアウトから生成されたアダプタオブジェクトを使用してリッチ GUI I/O 処理を実行します。
PROCESS PAGE UPDATE	PROCESS PAGE ステートメントを再実行します。
PROCESS PAGE MODAL	処理ブロックを開始し、リッチ GUI ウィンドウの存続期間を制御します。

論理条件の処理

Natural プログラムの実行中に検出された条件に基づいてステートメントの実行を制御するには、次のステートメントを使用します。

IF	論理条件に基づいてステートメントを実行します。
IF SELECTION	一連の英数字フィールドのうちの 1 つにのみ値が含まれていることを確認します。
DECIDE FOR	複数の論理条件に基づいてステートメントを実行します。
DECIDE ON	変数の内容に基づいてステートメントを実行します。
ON ERROR	ランタイムエラーをインターセプトします。インターセプトしない場合は、Natural エラーメッセージが表示された後、Natural プログラムが終了します。

プログラムおよびルーチンの呼び出し

プログラムおよびルーチンの実行と関連して次のステートメントを使用します。

CALL	Natural プログラムから Natural 以外のプログラムを呼び出します。
CALLNAT	Natural サブプログラムを呼び出します。
CALL FILE	Natural 以外のプログラムを呼び出し、Adabas 以外のファイルからレコードを読み取ります。
CALL LOOP	Natural 以外のプログラムへのコールを含む処理ループを生成します。
DEFINE SUBROUTINE	Natural サブルーチンを定義します。
ESCAPE	ルーチンの実行を中止します。
FETCH	Natural プログラムを呼び出します。
PERFORM	Natural サブルーチンを呼び出します。
PROCESS COMMAND	コマンドプロセッサを呼び出します。
RUN	ソースプログラムをコンパイルして実行します。
ファンクションコール	Natural 機能を呼び出します。

プログラムとセッションの終了

アプリケーションの実行またはNaturalセッションを終了するには、次のNaturalステートメントを使用します。

STOP	アプリケーションの実行を終了します。
TERMINATE	Natural セッションを終了します。

ワークファイルの制御

（Adabas以外の）物理順ワークファイルに対するデータの読み取り／書き込みには次のNaturalステートメントを使用します。

WRITE WORK FILE	ワークファイルにデータを書き込みます。
READ WORK FILE	ワークファイルからデータを読み取ります。
CLOSE WORK FILE	ワークファイルをクローズします。
DEFINE WORK FILE	ファイル名をワークファイルに割り当てます。

コンポーネントベースプログラミング

コンポーネントベースプログラミングと関連して次の Natural ステートメントを使用します。

DEFINE CLASS	Natural クラスモジュール内からクラスを指定します。
CREATE OBJECT	指定したクラスのオブジェクト（インスタンスとも呼ばれる）を作成します。
SEND METHOD	オブジェクトのメソッドを呼び出します。
INTERFACE	クラスの特定機能に対するインターフェイス（メソッドおよびプロパティの集合）を定義します。
METHOD	インターフェイス定義外のメソッドの実装として、サブプログラムを割り当てます。
PROPERTY	インターフェイス定義外のプロパティに対する実装として、オブジェクトデータ変数を割り当てます。

イベントドリブンプログラミング

イベントドリブンプログラミングには次の Natural ステートメントを使用します。

OPEN DIALOG	ダイアログを開きます。
CLOSE DIALOG	ダイアログを閉じます。
SEND EVENT	ユーザー定義イベントを起動します。
PROCESS GUI	イベントドリブンアプリケーションの標準プロシージャを実行します。

ダイナミック変数または X-array のメモリ管理制御

EXPAND	ダイナミック変数の割り当てメモリを指定したサイズに拡張します。または、X-array のオカレンス数を拡張します。
REDUCE	ダイナミック変数のサイズまたは X-array のオカレンス数を縮小します。
RESIZE	ダイナミック変数のサイズまたは X-array のオカレンス数を調整します。

インターネットと XML

PARSE	Natural プログラムから XML ドキュメントを解析できるようになります。
REQUEST DOCUMENT	外部システムへのアクセスを可能にします。

その他

DEFINE DATA	Natural プログラムまたはルーチンで使用するデータ要素を定義します。
END	Natural プログラムまたはルーチンのソースコードの終了を示します。
INCLUDE	コンパイル時に Natural コピーコードを組み込みます。
PROCESS REPORTER	プログラム内から Natural Reporter と通信し、特定のアクションを実行するように Reporter に指示します。
RELEASE	Natural スタックの内容を削除します。FIND ステートメントによって保存された ISN の集合を解放します。Natural グローバル変数を解放します。
SET CONTROL	Natural プログラム内から Natural 端末コマンドを実行します。
SET KEY	端末キーに機能を割り当てます。

SET TIME	*TIMD システム変数に対する参照開始時点を設定します。
STACK	Natural スタックにデータ／コマンドを挿入します。

レポーティングモードのステートメント

次のステートメントは、レポーティングモードでのみ使用できます。

CLOSE LOOP	処理ループを閉じます。
DO/DOEND	論理条件に基づいて実行するステートメントのグループを指定します。
OBTAIN	ファイルから 1 つ以上のフィールドを読み取ります。
REDEFINE	フィールドを再定義します。
SET GLOBALS	セッションパラメータの値を設定します。

次のステートメントは、ストラクチャードモードとレポーティングモードの両方で使用できます。ただし、ステートメント構造と機能が異なるものがあります。

AT START OF DATA	処理ループで一連のレコードの先頭が処理されるときに実行されるステートメントを指定します。
AT END OF DATA	処理ループで一連のレコードの最後が処理されたときに実行されるステートメントを指定します。
AT BREAK	制御フィールドの値が変更されるときに実行されるステートメントを指定します（ブレイク処理）。
AT TOP OF PAGE	新しい出力ページの開始時に実行される処理を指定します。
AT END OF PAGE	出力ページの最後に達したときに実行される処理を指定します。
BEFORE BREAK PROCESSING	ブレイク処理の実行前に実行されるステートメントを指定します。
CALL LOOP	Natural 以外のプログラムへのコールを含む処理ループを生成します。
CALL FILE	Natural 以外のプログラムを呼び出し、Adabas 以外のファイルからレコードを読み取ります。
COMPUTE	算術演算を実行するか、値をフィールドに割り当てます。
DEFINE SUBROUTINE	Natural サブルーチンを定義します。
ESCAPE	処理ループの実行を停止します。
FIND	ユーザー指定の条件に基づいて、データベースファイルからレコードを選択します。
GET SAME	現在処理中のレコードを再読み取りします。
HISTOGRAM	データベースフィールドの値を読み取ります。
IF	論理条件に基づいてステートメントを実行します。

IF SELECTION	一連の英数字フィールドのうちの 1 つにのみ値が含まれていることを確認します。
ON ERROR	ランタイムエラーをインターセプトします。インターセプトしない場合は、Natural エラーメッセージが表示された後、Natural プログラムが終了します。
PARSE	Natural プログラムから XML ドキュメントを解析できるようになります。
READ	データベースファイル内の物理的または論理的な一連のレコードを読み取ります。
READ WORK FILE	ワークファイルからデータを読み取ります。
REPEAT	処理ループを開始し、指定された条件に基づいて終了します。
SORT	レコードをソートします。
STORE	新しいレコードをデータベースに追加します。
UPDATE	データベース内のレコードを更新します。

5 ACCEPT/REJECT

```
{ ACCEPT } [IF] logical-condition
{ REJECT }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **HISTOGRAM** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

ステートメント ACCEPT および REJECT は、ユーザー指定の論理条件を基準にして、レコードを受け入れまたは除外します。 ACCEPT/REJECT ステートメントは、データレコードを読み込むステートメント（FIND、READ、HISTOGRAM、CALL FILE、SORT、READ WORK FILE）とともに、処理ループ内で使用します。 指定した条件は、レコードが選択され、読み込まれた後に評価されます。

ACCEPT/REJECT ステートメントの処理に達すると、レコードを読むステートメントで開始された現在稼働している最も内側の処理ループが、内部的に参照されます。

ACCEPT/REJECT ステートメントがサブルーチン内にあり、それがレコードの除外（REJECT）である場合、処理ループに入ったサブルーチンは自動的に終了します。そして、現在稼働している最も内側の処理ループの次のレコードで処理が続けられます。

構文説明

IF	FIND、READ、または HISTOGRAM ステートメントでレコードが選択され、読み込まれる場合、IF 節を ACCEPT または REJECT ステートメントで使用して、指定された条件に加えて論理条件の基準を指定することもできます。論理条件の基準は、レコードが読み込まれ、レコードの処理が開始した後に評価されます。
logical-condition	基本の条件は、1 つの関係式です。 複数の関係式を論理演算子（AND、OR）と組み合わせて、複合条件を構成することができます。 また、演算式を使用して、1 つの関係式を構成することもできます。 論理条件に使用するフィールドとして、データベースフィールドあるいはユーザー定義変数が指定できます。 論理条件の詳細については、『*プログラミングガイド*』の「*論理条件基準*」を参照してください。

	ACCEPT/REJECT ステートメントを HISTOGRAM ステートメントで使用する場合、HISTOGRAM ステートメントで指定したデータベースフィールドだけが論理条件として使用できます。

複数の ACCEPT/REJECTステートメントの処理

通常 1 つの処理ループ内では、ACCEPT または REJECT ステートメントを使用します。 2 つ以上の ACCEPT/REJECT を*連続*して指定した場合は、次の条件が適用されます。

- 同一処理ループ内に、続けて ACCEPT および REJECT ステートメントを使用すると、指定した順序で処理されます。
- ACCEPT 条件を満たすと、そのレコードは受け入れ（ACCEPT）られます。同じ処理ループ内の連続した ACCEPT/REJECT ステートメントは処理されません。
- REJECT 条件を満たすと、そのレコードは除外（REJECT）されます。同じ処理ループ内の連続した ACCEPT/REJECT ステートメントは処理されません。
- 処理が最後の ACCEPT/REJECT ステートメントに達すると、そのレコードを受け入れるかどうかは、最後の ACCEPT/REJECT ステートメントで決定されます。

他のステートメントが、複数の ACCEPT/REJECT ステートメントの間に挿入されていると、各 ACCEPT/REJECT ステートメントは独立して処理されます。

リミット表記

ACCEPT または REJECT ステートメントを含む処理ループに対して、LIMIT ステートメントや他の制限指定が与えられている場合は、各処理レコードは受け入れられる（ACCEPT）か除外される（REJECT）かにかかわらずカウントされ、制限数と比較されます。

例

- 例 1 - ACCEPT

例 1 - ACCEPT

```
** Example 'ACREX1': ACCEPT
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 SEX
  2 MAR-STAT
END-DEFINE
*
LIMIT 50
READ EMPLOY-VIEW
  ACCEPT IF SEX='M' AND MAR-STAT = 'S'
  WRITE  NOTITLE '=' NAME '=' SEX 5X '=' MAR-STAT
END-READ
END
```

プログラム **ACREX1** の出力：

```
NAME: MORENO               S E X: M     MARITAL STATUS: S
NAME: VAUZELLE             S E X: M     MARITAL STATUS: S
NAME: BAILLET              S E X: M     MARITAL STATUS: S
NAME: HEURTEBISE           S E X: M     MARITAL STATUS: S
NAME: LION                 S E X: M     MARITAL STATUS: S
NAME: DEZELUS              S E X: M     MARITAL STATUS: S
NAME: BOYER                S E X: M     MARITAL STATUS: S
NAME: BROUSSE              S E X: M     MARITAL STATUS: S
NAME: DROMARD              S E X: M     MARITAL STATUS: S
NAME: DUC                  S E X: M     MARITAL STATUS: S
NAME: BEGUERIE             S E X: M     MARITAL STATUS: S
NAME: FOREST               S E X: M     MARITAL STATUS: S
NAME: GEORGES              S E X: M     MARITAL STATUS: S
```

例 2 - ACCEPT / REJECT

```
** Example 'ACREX2': ACCEPT/REJECT
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 SALARY    (1)
*
1 #PROC-COUNT (N8) INIT <0>
```

```
END-DEFINE
*
EMP. FIND EMPLOY-VIEW WITH NAME = 'JACKSON'
  WRITE NOTITLE *COUNTER NAME FIRST-NAME 'SALARY:' SALARY(1)
  /*
  ACCEPT IF SALARY (1) LT 50000
  WRITE *COUNTER 'ACCEPTED FOR FURTHER PROCESSING'
  /*
  REJECT IF SALARY (1) GT 30000
  WRITE *COUNTER 'NOT REJECTED'
  /*
  ADD 1 TO #PROC-COUNT
END-FIND
*
SKIP 2
WRITE NOTITLE 'TOTAL PERSONS FOUND ' *NUMBER (EMP.) /
              'TOTAL PERSONS SELECTED' #PROC-COUNT
END
```

プログラム **ACREX2** の出力：

```
          1 JACKSON              CLAUDE               SALARY:      33000
          1 ACCEPTED FOR FURTHER PROCESSING
          2 JACKSON              FORTUNA              SALARY:      36000
          2 ACCEPTED FOR FURTHER PROCESSING
          3 JACKSON              CHARLIE              SALARY:      23000
          3 ACCEPTED FOR FURTHER PROCESSING
          3 NOT REJECTED


TOTAL PERSONS FOUND           3
TOTAL PERSONS SELECTED        1
```

6 ADD

ADD ステートメントを使用して、複数のオペランドを追加します。

このchapterでは、次のトピックについて説明します。

関連ステートメント：**COMPRESS** | **COMPUTE** | **DIVIDE** | **EXAMINE** | **MOVE** | **MOVE ALL** | **MULTIPLY** | **RESET** | **SEPARATE** | **SUBTRACT**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

ADD ステートメントを使用して、複数のオペランドを追加します。

Notes:

1. ADD ステートメント実行時、算術演算に使用される各オペランドは、有効な値を持っている必要があります。
2. 配列を含む加算については、「*配列の算術演算*」を参照してください。
3. オペランドのフォーマットについては、「*フォーマット混合式のパフォーマンスについて*」を参照してください。

構文説明

このステートメントには、2 つの異なる構造が可能です。

- **構文 1**
- **構文 2**

構文図で使用されている記号については、「*構文記号*」を参照してください。

構文 1

```
ADD [ROUNDED] operand1... TO operand2
```

オペランド定義テーブル（構文 1）：

オペランド	構文要素								フォーマット										ステートメント参照	ダイナミック定義
operand1	C	S	A		N				N	P	I	F		D	T				可	不可
operand2		S	A		M				N	P	I	F		D	T				可	可

構文要素の説明：

operand1	*operand1* は加数です。
ROUNDED	キーワード ROUNDED を使用すると、結果が四捨五入されます。 四捨五入のルールについては、「*演算割り当てのルール*」を参照してください。
TO *operand2*	*operand2* は、結果フィールドです。 結果フィールドも加算され、加算結果もこの結果フィールドに保存されます。

構文 2

```
ADD [ROUNDED] operand1... GIVING operand2
```

オペランド定義テーブル（構文 2）：

オペランド	構文要素						フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S	A		N				N	P	I	F		D	T				可	不可
operand2		S	A		M		A	U	N	P	I	F	B*	D	T				可	可

**operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

operand1	*operand1* は加数です。
ROUNDED	キーワード ROUNDED を使用すると、結果が四捨五入されます。 四捨五入のルールについては、「*演算割り当てのルール*」を参照してください。
GIVING *operand2*	*operand2* は結果フィールドですが、構文 1 とは異なり、結果の*保存*のみに使用されます。 *operand2* を英数字フィーマットで定義した場合、結果は英数字フォーマットに変換されます。 データベースフィールドを結果フィールドとして使用しても、ADD演算は、プログラム内で使用する内部値だけを更新します。 データベースのフィールド値は変わりません。

例

```
** Example 'ADDEX1': ADD
************************************************************************
DEFINE DATA LOCAL
1 #A      (P2)
1 #B      (P1.1)
1 #C      (P1)
1 #DATE   (D)
1 #ARRAY1 (P5/1:4,1:4) INIT (2,*) <5>
1 #ARRAY2 (P5/1:4,1:4) INIT (4,*) <10>
END-DEFINE
*
ADD +5 -2 -1 GIVING #A
WRITE NOTITLE 'ADD +5 -2 -1 GIVING #A' 15X '=' #A
*
ADD .231 3.6 GIVING #B
WRITE       / 'ADD .231 3.6 GIVING #B' 15X '=' #B
*
ADD ROUNDED 2.9 3.8 GIVING #C
WRITE       / 'ADD ROUNDED 2.9 3.8 GIVING #C' 8X '=' #C
*
MOVE *DATX TO #DATE
ADD 7 TO #DATE
WRITE       / 'CURRENT DATE:'     *DATX (DF=L) 13X
              'CURRENT DATE + 7:' #DATE (DF=L)
*
WRITE       / '#ARRAY1 AND #ARRAY2 BEFORE ADDITION'
            / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
ADD #ARRAY1 (2,*) TO #ARRAY2 (4,*)
WRITE       / '#ARRAY1 AND #ARRAY2 AFTER ADDITION'
            / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
*
END
```

プログラム **ADDEX1** の出力：

```
ADD +5 -2 -1 GIVING #A               #A:   2

ADD .231 3.6 GIVING #B               #B:  3.8

ADD ROUNDED 2.9 3.8 GIVING #C        #C:  7

CURRENT DATE: 2005-01-10             CURRENT DATE + 7: 2005-01-17

#ARRAY1 AND #ARRAY2 BEFORE ADDITION
#ARRAY1:      5      5      5      5 #ARRAY2:     10     10     10     10
```

```
#ARRAY1 AND #ARRAY2 AFTER ADDITION
#ARRAY1:      5      5      5      5 #ARRAY2:     15     15     15     15
```

7 ASSIGN

COMPUTE ステートメントを参照してください。

8 AT BREAK

ストラクチャードモード構文

```
[AT] BREAK [(r)] [OF] operand1 [/n/]
statement ...
END-BREAK
```

レポーティングモード構文

```
[AT] BREAK [(r)] [OF] operand1 [/n/]
          { statement            }
          { DO statement... DOEND }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント： **ACCEPT/REJECT** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

AT BREAK ステートメントは、コントロールフィールドの値が変わるたびに、1 つ以上のステートメントを実行します。このステートメントは、自動ブレイク処理と組み合わせて使用します。また、FIND、READ, HISTOGRAM、SORT、READ WORK FILE ステートメントとともに使用します。

AT BREAK ステートメントブロックは、ブレイク条件の発生時にそのステートメントを含んだオブジェクトがアクティブな場合にだけ実行されます。

AT BREAK 条件内で新しく処理ループを開始することができます。 ただし、このループは、その AT BREAK 条件内で閉じる必要があります。

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

AT BREAK ステートメントで使用できる Natural システム関数については、『*システム関数*』ドキュメントの「*処理ループで使用する Natural システム関数*」および『*プログラミングガイド*』の「*AT BREAK ステートメントとシステム関数の例*」を参照してください。

詳細については、『*プログラミングガイド*』の「*AT BREAK ステートメント*」も参照してください。 次のトピックについて説明します。

- *データベースフィールドに基づくコントロールブレイク*
- *ユーザー定義変数に基づくコントロールブレイク*
- *自動ブレイク処理*

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1		S				A	U	N	P	I	F	B	D	T	L				可	不可

構文要素の説明：

(*r*)	参照表記： デフォルトで、最終の AT BREAK 条件（ループ終了処理）は、常に **FIND**、**READ**、**READ WORK FILE**、**HISTOGRAM**、または **SORT** ステートメントで開始された、一番外側で稼働している処理ループに関連します。 表記（*r*）を使用すると、AT BREAK ステートメントの最終ブレイクを、現在オープンしている特定の処理ループに関連付けることができます（AT BREAKステートメントが含まれるそのループを含めて外側のどのループにでも関連付けられます）。 例： `0110 ...` `0120 READ ...` `0130   FIND ...` `0140     FIND ...` `0150       AT BREAK ...` `0160         FIND ...` `0170         END-FIND` `0180       END-BREAK` `0190     END-FIND` `0200   END-FIND` `0210 END-READ` `0220 ...` この例では、最終の AT BREAK 条件は、行番号 0120 で開始する READ ループに関連付けられます。 また、AT BREAK 条件を行番号 0130 や 0140 で開始する FIND ループに関連付けられます（ただし、行番号 0160 の FIND には関連付けられません）。 ブレイク階層に "(*r*)" を指定する場合は、最初の AT BREAK ステートメントで指定する必要があります。この指定は、その後の全 AT BREAK ステートメントにも適用されます。

operand1	コントロールフィールド： ブレイクコントロールフィールドとして使用されるフィールドで、通常はデータベースフィールドです。ユーザー定義変数を使用する場合は、自動ブレイク処理の評価の前に初期化されている必要があります（BEFORE BREAK PROCESSING ステートメントを参照）。 配列の特定のオカレンスをコントロールフィールドとして使用することもできます。
*l*n*l*	-表記 /*n*/ を使用すると、値の変化を調べるためにチェックされるのは、コントロールフィールドの（左から右へ数えて）最初の *n* 個の位置だけであることを示すことができます。 この表記は、オペランドがフォーマット A、B、N、または P の場合に使用できます。 コントロールブレイクは、コントロールフィールドの値が変わるか、または AT BREAK ステートメントを適用する処理ループですべてのレコードの処理が終了したときに発生します。
END-BREAK	Natural 予約語 END-BREAK を使用して、AT BREAK ステートメントを終了させる必要があります。

複数ブレイクレベルの階層

同一プログラムモジュール内の同一処理ループに、複数の AT BREAK ステートメントを指定することができます。 同一処理ループに指定した複数の AT BREAK ステートメントは、連続指定しても、間に他のステートメントを指定しても、ブレイクレベルの階層を形成します。 最初の AT BREAK ステートメントは、一番低いレベルのコントロールブレイクです。続く各 AT BREAK ステートメントは、コントロールブレイクの階層でそれぞれ次に高いブレイクレベルとなります。

階層構造を作る処理ループでも、それぞれのループ内に AT BREAK ステートメントの階層を持つことができます。

例：

ストラクチャードモード：

```
FIND ...
   AT BREAK
   ...
   END-BREAK
   AT BREAK
   ...
   END-BREAK
   AT BREAK
   ...
   END-BREAK
END-FIND
...
```

レポーティングモード：

```
FIND ...
   AT BREAK
      DO
      ...
      DOEND
   AT BREAK
      DO
      ...
      DOEND
...
```

あるブレイクレベルのコントロールフィールドの値が変わると、そのブレイクレベルと、より低いブレイクレベルの処理がすべて行われます（より低いブレイクレベルのコントロールフィールドの値とは関係なく行われます）。

複数のブレイク処理を連続して指定すると、プログラムメンテナンスが容易になります。

下記の「*例3*」および『*プログラミングガイド*』の「*複数のコントロールブレイクレベル*」セクションを参照してください。

例

このsectionでは、次のトピックについて説明します。

- 例 1 - AT BREAK
- 例 2 - *In* 表記を使用した AT BREAK
- 例 3 - AT BREAK による複数ブレイクレベルの指定

AT BREAK の他の例については、「処理ループで使用する Natural システム関数」、および例 ATBEX3 と ATBEX4 を参照してください。

例 1 - AT BREAK

```
** Example 'ATBEX1S': AT BREAK (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
  2 NAME
END-DEFINE
*
LIMIT 10
READ EMPLOY-VIEW BY CITY
AT BREAK OF CITY
    SKIP 1
  END-BREAK
  DISPLAY NOTITLE CITY (IS=ON) COUNTRY (IS=ON) NAME
END-READ
*
END
```

プログラム **ATBEX1S** の出力：

```
        CITY         COUNTRY         NAME
-------------------- ------- --------------------

AIKEN                USA     SENKO

AIX EN OTHE          F       GODEFROY

AJACCIO                      CANALE

ALBERTSLUND          DK      PLOUG

ALBUQUERQUE          USA     HAMMOND
                             ROLLING
                             FREEMAN
                             LINCOLN

ALFRETON             UK      GOLDBERG

ALICANTE             E       GOMEZ
```

レポーティングモードの例については、ライブラリ SYSEXRM のプログラム **ATBEX1R** を参照してください。

例 2 - *lnl* 表記を使用した AT BREAK

```
** Example 'ATBEX2': AT BREAK (with /n/ notation)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 DEPT
  2 NAME
END-DEFINE
*
LIMIT 10
READ EMPLOY-VIEW BY DEPT STARTING FROM 'A'
AT BREAK OF DEPT /4/
    SKIP 1
  END-BREAK
  DISPLAY NOTITLE DEPT NAME
END-READ
*
END
```

プログラム **ATBEX2** の出力：

```
DEPARTMENT         NAME
   CODE
---------- --------------------

ADMA01     JENSEN
ADMA01     PETERSEN
ADMA01     MORTENSEN
ADMA01     MADSEN
ADMA01     BUHL
ADMA02     HERMANSEN
ADMA02     PLOUG
ADMA02     HANSEN

COMP01     HEURTEBISE
COMP01     TANCHOU
```

例 3 - AT BREAK による複数ブレイクレベルの指定

```
** Example 'ATBEX5S': AT BREAK (multiple break levels) (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 DEPT
  2 NAME
  2 LEAVE-DUE
1 #LEAVE-DUE-L (N4)
END-DEFINE
*
LIMIT 5
FIND EMPLOY-VIEW WITH CITY = 'PHILADELPHIA' OR = 'PITTSBURGH'
                 SORTED BY CITY DEPT
  MOVE LEAVE-DUE TO #LEAVE-DUE-L
  DISPLAY CITY (IS=ON) DEPT (IS=ON) NAME #LEAVE-DUE-L
  /*
 AT BREAK OF DEPT
    WRITE NOTITLE /
          T*DEPT OLD(DEPT) T*#LEAVE-DUE-L SUM(#LEAVE-DUE-L) /
  END-BREAK
  AT BREAK OF CITY
    WRITE NOTITLE
          T*CITY OLD(CITY) T*#LEAVE-DUE-L SUM(#LEAVE-DUE-L) //
  END-BREAK
END-FIND
*
END
```

プログラム **ATBEX5** の出力：

```
        CITY          DEPARTMENT          NAME          #LEAVE-DUE-L
                         CODE
-------------------- ---------- -------------------- ------------

PHILADELPHIA         MGMT30     WOLF-TERROINE            11
                                MACKARNESS               27

                     MGMT30                              38

                     TECH10     BUSH                     39
                                NETTLEFOLDS              24

                     TECH10                              63

PHILADELPHIA                                            101


PITTSBURGH           MGMT10     FLETCHER                 34

                     MGMT10                              34

PITTSBURGH                                               34
```

レポーティングモードの例については、ライブラリ SYSEXRM のプログラム **ATBEX5R** を参照してください。

9 AT END OF DATA

ストラクチャードモード構文

```
[AT]END [OF] DATA [(r)]
statement ...
END-ENDDATA
```

レポーティングモード構文

```
[AT]END [OF] DATA [(r)]
{     statement          }
{     DO statement ... DOEND }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

`AT END OF DATA` ステートメントは、データベース処理ループで選択された全レコードの処理が終了したときに行う処理を指定します。

このsectionでは、次のトピックについて説明します。

- **処理**
- **データベースフィールドの値**
- **位置**
- **システム関数**

『*プログラミングガイド*』の「*AT START/END OF DATA ステートメント*」を参照してください。

処理

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

データベースフィールドの値

処理ループで AT END OF DATA 条件が発生したときに、すべてのデータベースフィールドは、最後に処理されたレコードの値になります。

位置

このステートメントは、対応するループ生成ステートメントと同じプログラムモジュール内に指定する必要があります。

システム関数

AT END OF DATA ステートメントで使用できる Natural システム関数については、『*システム関数*』ドキュメントの「*処理ループでのシステム関数の使用*」セクションを参照してください。

制限事項

- このステートメントは、FIND、READ、READ WORK FILE、HISTOGRAM、または SORT ステートメントで始まる処理ループだけに使用できます。
- また、処理ループ内で 1 回だけ使用できます。
- END OF DATAが参照する処理ループに入らなかった場合、このステートメントは*評価されません*。

構文説明

(*r*)	特定の処理ループの参照： AT END OF DATA ステートメントは、表記 (r) によって、アクティブな特定の処理ループに関連付けることができます。 この表記がない場合、AT END OF DATA ステートメントはアクティブなデータベース処理ループ内の一番外側に関連付けられます。
END-ENDDATA	Natural 予約語 END-ENDDATA を使用して、AT END OF DATA ステートメントを終了させる必要があります。

例

```
** Example 'AEDEX1S': AT END OF DATA
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 5
EMP. FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
  IF NO RECORDS FOUND
    ENTER
  END-NOREC
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
          SALARY (1) CURR-CODE (1)
  /*
   AT END OF DATA
    IF *COUNTER (EMP.) = 0
      WRITE 'NO RECORDS FOUND'
      ESCAPE BOTTOM
    END-IF
    WRITE NOTITLE / 'SALARY STATISTICS:'
                  / 7X 'MAXIMUM:' MAX(SALARY(1))  CURR-CODE (1)
                  / 7X 'MINIMUM:' MIN(SALARY(1))  CURR-CODE (1)
                  / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
    END-ENDDATA
  /*
END-FIND
*
END
```

「*処理ループで使用する Natural システム関数*」も参照してください。

プログラム **AEDEX1S** の出力：

```
PERSONNEL         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     CODE
--------- -------------------- -------------------- ---------- --------

11100328  BERGHAUS             ROSE                    70800 DM
11100329  BARTHEL              PETER                   42000 DM
11300313  AECKERLE             SUSANNE                 55200 DM
11300316  KANTE                GABRIELE                61200 DM
11500304  KLUGE                ELKE                    49200 DM

SALARY STATISTICS:
       MAXIMUM:      70800 DM
       MINIMUM:      42000 DM
       AVERAGE:      55680 DM
```

レポーティングモードの例については、ライブラリ SYSEXRM のプログラム **AEDEX1R** を参照してください。

10 AT END OF PAGE

ストラクチャードモード構文

```
[AT] END [OF] PAGE [(rep)]
statement ...
END-ENDPAGE
```

レポーティングモード構文

```
[AT] END [OF] PAGE [(rep)]
{     statement           }
{     DO statement ... DOEND }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

AT END OF PAGE ステートメントは、END OF PAGE（ページ終了）条件が発生したときに行う処理を指定します（『パラメータリファレンス』ドキュメントのセッションパラメータ PS を参照）。 END OF PAGE 条件は、SKIP または NEWPAGE ステートメントでも発生します。ただし、EJECT または INPUT ステートメントの場合は発生しません。

『*プログラミングガイド*』の次のセクションも参照してください。

- *データ出力制御*
- *レポート指定 - (rep) 表記*
- *出力ページのレイアウト*
- *AT END OF PAGE ステートメント*

処理

AT END OF PAGE ステートメントブロックは、END OF PAGE 条件が発生したときにそのステートメントブロックを含むオブジェクトが、アクティブな場合にのみ実行されます。

AT END OF PAGE ステートメントは、内部サブルーチン内には指定できません。

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

論理ページサイズ

END OF PAGE のチェックは、DISPLAY や WRITE ステートメントの処理の完了後に実行されます。 そのため、DISPLAY や WRITE ステートメントで複数行の出力を行うと、END OF PAGE 条件が発生する前に、物理ページのオーバーフローを起こすことがあります。

AT END OF PAGE ステートメントでプリントされる情報が、タイトルと同じ物理ページに出力されることを保証するには、物理的なページサイズより小さい論理ページサイズ（セッションパラメータ PS）を指定する必要があります。

最終ページの扱い

メインプログラム内では、ESCAPE、STOP、または END によりメインプログラムの実行が終了すると、END OF PAGE 条件が発生します。

サブルーチン内では、ESCAPE-ROUTINE、RETURN、または END-SUBROUTINE によりサブルーチンの実行が終了すると、END OF PAGE 条件が発生します。

システム関数

AT END OF PAGE ステートメントで使用できる Natural システム関数については、『*システム関数*』ドキュメントの「*処理ループでのシステム関数の使用*」セクションを参照してください。

AT END OF PAGE ステートメントブロック内で、システム関数を使用する場合は、GIVE SYSTEM FUNCTIONS 節を、対応する DISPLAY ステートメントで指定する必要があります。

AT END OF PAGE と INPUT ステートメント

AT END OF PAGE ステートメントブロック内に、INPUT ステートメントを指定すると、NEWPAGE（ページ換え）処理は実行されません。 INPUT ステートメントで作成した行を同じ物理ページに表示するために、セッションパラメータ PS でページサイズを物理ページサイズより小さい値に指定する必要があります。

以下の項目も参照してください。

- INPUT ステートメントの*画面分割機能*

構文説明

(*rep*)	レポート指定： 表記 (*rep*) を使用すると、AT END OF PAGE ステートメントを適用するレポートの ID を指定できます。 範囲 0～31 の値、または **DEFINE PRINTER** ステートメントを使用して割り当てた論理名を指定できます。 (*rep*) を指定しない場合、AT END OF PAGE ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成した出力レポートのフォーマットを制御する方法については、『*プログラミングガイド*』の「*データ出力の制御*」を参照してください。
END-ENDPAGE	Natural 予約語 END-ENDPAGE を使用して、AT END OF PAGE ステートメントを終了させる必要があります。

例

例 1 - AT END OF PAGE

```
** Example 'AEPEX1S': AT END OF PAGE (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
FORMAT PS=10
LIMIT 10
READ EMPLOY-VIEW BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
          NAME JOB-TITLE 'SALARY' SALARY(1) CURR-CODE (1)
  /*

 AT END OF PAGE
```

```
    WRITE / 28T 'AVERAGE SALARY: ...' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDPAGE


END-READ
*
END
```

「*処理ループで使用する Natural システム関数*」も参照してください。

プログラム **AEPEX1S** の出力：

```
        NAME                  CURRENT            SALARY   CURRENCY
                             POSITION                       CODE
-------------------- ------------------------ ---------- --------

CREMER               ANALYST                       34000 USD
MARKUSH              TRAINEE                       22000 USD
GEE                  MANAGER                       39500 USD
KUNEY                DBA                           40200 USD
NEEDHAM              PROGRAMMER                    32500 USD
JACKSON              PROGRAMMER                    33000 USD

                           AVERAGE SALARY: ...     33533 USD
```

レポーティングモードの例については、ライブラリ SYSEXRM のプログラム **AEPEX1R** を参照してください。

例 2 - INPUT ステートメントでの AT END OF PAGE

```
** Example 'AEPEX2': AT END OF PAGE (with INPUT)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 POST-CODE
  2 CITY
*
1 #START-NAME (A20)
END-DEFINE
*
FORMAT PS=21
*
REPEAT
  READ (15) EMPLOY-VIEW BY NAME = #START-NAME
    DISPLAY NOTITLE NAME FIRST-NAME POST-CODE CITY
  END-READ
```

```
  NEWPAGE
  /*
 AT END OF PAGE
    MOVE NAME TO #START-NAME
    INPUT / '-' (79)
          / 10T 'Reposition to name ==>'
                #START-NAME (AD=MI) '(''.'' to exit)'
    IF #START-NAME = '.'
      STOP
    END-IF
  END-ENDPAGE
  /*
END-REPEAT
END
```

プログラム **AEPEX2S** の出力：

```
        NAME              FIRST-NAME        POSTAL           CITY
                                           ADDRESS
-------------------- -------------------- ---------- --------------------

ABELLAN              KEPA                 28014      MADRID
ACHIESON             ROBERT               DE3 4TR    DERBY
ADAM                 SIMONE               89300      JOIGNY
ADKINSON             JEFF                 11201      BROOKLYN
ADKINSON             PHYLLIS              90211      BEVERLEY HILLS
ADKINSON             HAZEL                20760      GAITHERSBURG
ADKINSON             DAVID                27514      CHAPEL HILL
ADKINSON             CHARLIE              21730      LEXINGTON
ADKINSON             MARTHA               17010      FRAMINGHAM
ADKINSON             TIMMIE               17300      BEDFORD
ADKINSON             BOB                  66044      LAWRENCE
AECKERLE             SUSANNE              7000       STUTTGART
AFANASSIEV           PHILIP               39401      HATTIESBURG
AFANASSIEV           ROSE                 60201      EVANSTON
AHL                  FLEMMING             2300       SUNDBY

--------------------------------------------------------------------------------
         Reposition to name ==> AHL                  ('.' to exit)
```

11 AT START OF DATA

ストラクチャードモード構文

```
[AT] START [OF] DATA [(r)]
statement ...
END-START
```

レポーティングモード構文

```
[AT] START [OF] DATA [(r)]
{       statement           }
{       DO statement... DOEND }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

AT START OF DATA ステートメントは、READ、FIND、HISTOGRAM、SORT、または READ WORK FILE ステートメントで始まる処理ループで最初のレコードが読まれた直後の処理を指定します。

『*プログラミングガイド*』の「*AT START/END OF DATA ステートメント*」を参照してください。

処理

ループ開始ステートメントに WHERE 節がある場合は、基本検索条件と WHERE 条件の両方を満たす最初のレコードが読まれた時点で、AT START OF DATA 条件が真になります。

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

データベースフィールドの値

データベースフィールドには、AT START OF DATA 条件が真になるレコードの値が含まれます。つまり、処理するすべてのレコード集合の第 1 レコードの値になります。

位置

このステートメントは、処理ループの*中に*指定する必要があります。また、その処理ループに対して 1 回だけ使用できます。

構文説明

(*r*)	特定の処理ループの参照： AT START OF DATA ステートメントは、表記 (r) によって、アクティブな特定の処理ループに関連付けることができます。 この表記を使用しなかった場合、ステートメントはアクティブな処理ループ内の一番外側に関連付けられます。
END-START	Natural 予約語 END-START を使用して、AT START OF DATA ステートメントを終了させる必要があります。

例

```
** Example 'ASDEX1S': AT START OF DATA (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
*
1 #CNTL (A1)  INIT <' '>
1 #CITY (A20) INIT <' '>
END-DEFINE
*
REPEAT
  INPUT 'ENTER VALUE FOR CITY'  #CITY
  IF #CITY = ' ' OR = 'END'
    STOP
  END-IF
  FIND EMPLOY-VIEW WITH CITY = #CITY
    IF NO RECORDS FOUND
      WRITE NOTITLE NOHDR 'NO RECORDS FOUND'
      ESCAPE BOTTOM
```

```
    END-NOREC
    /*
   AT START OF DATA
      INPUT (AD=O) 'RECORDS FOUND' *NUMBER //
                   'ENTER ''D'' TO DISPLAY RECORDS' #CNTL (AD=A)
      IF #CNTL NE 'D'
        ESCAPE BOTTOM
      END-IF
    END-START
    /*
    DISPLAY NAME FIRST-NAME
  END-FIND
END-REPEAT
END
```

プログラム **ASDEX1S** の出力：

```
ENTER VALUE FOR CITY PARIS
```

市町村名を入力して確認した後：

```
RECORDS FOUND        26

ENTER 'D' TO DISPLAY RECORDS D
```

表示されるレコード：

```
        NAME              FIRST-NAME
-------------------- --------------------

MAIZIERE             ELISABETH
MARX                 JEAN-MARIE
REIGNARD             JACQUELINE
RENAUD               MICHEL
REMOUE               GERMAINE
LAVENDA              SALOMON
BROUSSE              GUY
GIORDA               LOUIS
SIECA                FRANCOIS
CENSIER              BERNARD
DUC                  JEAN-PAUL
CAHN                 RAYMOND
MAZUY                ROBERT
FAURIE               HENRI
VALLY                ALAIN
BRETON               JEAN-MARIE
GIGLEUX              JACQUES
KORAB-BRZOZOWSKI     BOGDAN
```

```
XOLIN                CHRISTIAN
LEGRIS               ROGER
VVVV
```

レポーティングモードの例については、ライブラリ SYSEXRM のプログラム **ASDEX1R** を参照してください。

12 AT TOP OF PAGE

ストラクチャードモード構文

```
[AT] TOP [OF] PAGE [(rep)]
statement ...
END-TOPPAGE
```

レポーティングモード構文

```
[AT] TOP [OF] PAGE [(rep)]
      { statement              }
      { DO statement ... DOEND }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

AT TOP OF PAGE ステートメントは、新しいページが始まるときの処理を指定します。

『*プログラミングガイド*』の次のセクションも参照してください。

- *データ出力制御*
- *レポート指定 - (rep) 表記*
- *出力ページのレイアウト*
- *AT TOP OF PAGE ステートメント*

処理

新しいページが始まるのは、内部の行カウンタがNaturalセッションパラメータ PS（Naturalレポートのページサイズ）で設定されたページサイズを超過したとき、あるいは NEWPAGE ステートメントが実行されたときです。 どちらの場合も、AT条件が真となります。 EJECT ステートメントは、新しいページを開始しますが、AT条件を発生させることはありません。

AT TOP OF PAGE ステートメントブロックは、AT条件が発生したときに、そのステートメントブロックを含むオブジェクトがアクティブな場合にのみ実行されます。

AT TOP OF PAGE 処理の結果の出力は、タイトル行と1行の空白行に続いて出力されます。

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

制限事項

AT TOP OF PAGE ステートメントは、内部サブルーチン内には指定できません。

構文説明

(*rep*)	レポート指定： 表記（*rep*）を使用すると AT TOP OF PAGE ステートメントを提供するレポートの ID を指定できます。 範囲 0～31 の値、または **DEFINE PRINTER** ステートメントを使用して割り当てた論理名を指定できます。 （*rep*）を指定しない場合、AT TOP OF PAGE ステートメントは最初のレポート（レポート0）に適用されます。 Natural で作成した出力レポートのフォーマットを制御する方法については、『*プログラミングガイド*』の「*データ出力の制御*」を参照してください。
END-TOPPAGE	Natural 予約語 END-TOPPAGE を使用して、AT TOP OF PAGE ステートメントを終了させる必要があります。

例

```
** Example 'ATPEX1S': AT TOP OF PAGE (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 DEPT
END-DEFINE
*
FORMAT PS=15
LIMIT 15
READ EMPLOY-VIEW BY NAME STARTING FROM 'L'
  DISPLAY 2X NAME 4X FIRST-NAME CITY DEPT
  WRITE TITLE UNDERLINED 'EMPLOYEE REPORT'
  WRITE TRAILER '-' (78)
  /*
 AT TOP OF PAGE
    WRITE 'BEGINNING NAME:' NAME
  END-TOPPAGE
  /*
  AT END OF PAGE
    SKIP 1
    WRITE 'ENDING NAME:   ' NAME
  END-ENDPAGE
END-READ
END
```

プログラム **ATPEX1S** の出力：

```
                                EMPLOYEE REPORT
------------------------------------------------------------------------------
BEGINNING NAME: LAFON
          NAME                 FIRST-NAME              CITY         DEPARTMENT
                                                                       CODE
  --------------------    -------------------- -------------------- ----------

  LAFON                   CHRISTIANE           PARIS                VENT18
  LANDMANN                HARRY                ESCHBORN             MARK29
  LANE                    JACQUELINE           DERBY                MGMT02
  LANKATILLEKE            LALITH               FRANKFURT            PROD22
  LANNON                  BOB                  LINCOLN              SALE20
  LANNON                  LESLIE               SEATTLE              SALE30
  LARSEN                  CARL                 FARUM                SYSA01
  LARSEN                  MOGENS               VEMMELEV             SYSA02
```

```
------------------------------------------------------------------------------

ENDING NAME:    LARSEN
```

レポーティングモードの例については、ライブラリ SYSEXRM のプログラム **ATPEX1R** を参照してください。

13 BACKOUT TRANSACTION

```
BACKOUT [TRANSACTION]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント： **ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

BACKOUT TRANSACTION ステートメントは、現在の論理トランザクション内で実行された、すべてのデータベース更新をバックアウトします。 このステートメントも、トランザクション中に保持されていたすべてのレコードを解放します。

BACKOUT TRANSACTION ステートメントは、Natural の制御下でデータベーストランザクションが発生した場合にのみ実行されます。 どのデータベースに対して、ステートメントが実行されるかは、Natural プロファイルパラメータ ET（END/BACKOUT TRANSACTION ステートメントの実行）の設定に依存します。

- ET=OFF の場合、トランザクションで影響を受けたデータベースに対してのみ、ステートメントが実行されます。
- ET=ON の場合、BACKOUT TRANSACTION または END TRANSACTION ステートメントの最後の実行以後に参照された全データベースに対して、ステートメントが実行されます。

Natural が発行する BACKOUT TRANSACTION

ユーザーが端末コマンド（コマンド %% または CLEAR キー）で現在の Natural を中断した場合、Natural によって BACKOUT TRANSACTION ステートメントが発行されます。

詳細については、『*端末コマンド*』ドキュメントで端末コマンド %% を参照してください。

追加情報

トランザクションバックアウト機能の使用に関する詳細については、『*プログラミングガイド*』の「*データベース更新- トランザクション処理*」セクションおよび「*トランザクションのバックアウト*」セクションを参照してください。

制限事項

このステートメントは Entire System Server では使用できません。

データベース固有の考慮事項

SQL データベース	ほとんどのSQLデータベースはワークの論理ユニットが終了するときにすべてのカーソルを閉じるため、BACKOUT TRANSACTION ステートメントをデータベースの更新処理ループ内に指定することはできません。このようなループの後に指定する必要があります。
XML データベース	BACKOUT TRANSACTION ステートメントをデータベースの更新処理ループ内に指定することはできません。このようなループの後に指定する必要があります。

例

```
** Example 'BOTEX1': BACKOUT TRANSACTION
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 DEPT
  2 LEAVE-DUE
  2 LEAVE-TAKEN
*
1 #DEPT (A6)
1 #RESP (A3)
END-DEFINE
*
LIMIT 3
INPUT 'DEPARTMENT TO BE UPDATED:' #DEPT
IF #DEPT = ' '
  STOP
```

```
END-IF
*
FIND EMPLOY-VIEW WITH DEPT = #DEPT
  IF NO RECORDS FOUND
    REINPUT 'NO RECORDS FOUND'
  END-NOREC
  INPUT 'NAME:       ' NAME (AD=O) /
        'LEAVE DUE:  ' LEAVE-DUE (AD=M) /
        'LEAVE TAKEN:' LEAVE-TAKEN (AD=M)
  UPDATE
END-FIND
*
INPUT 'UPDATE TO BE PERFORMED? YES/NO:' #RESP
DECIDE ON FIRST #RESP
  VALUE 'YES'
    END TRANSACTION
  VALUE 'NO'
    BACKOUT TRANSACTION
  NONE
    REINPUT 'PLEASE ENTER YES OR NO'
END-DECIDE
*
END
```

プログラム **BOTEX1** の出力：

```

DEPARTMENT TO BE UPDATED: MGMT30
```

MGMT30 部門の結果：

```

NAME:        POREE
LEAVE DUE:   45
LEAVE TAKEN: 31
```

確認照会：

```

UPDATE TO BE PERFORMED  YES/NO: NO
```

14 BEFORE BREAK PROCESSING

ストラクチャードモード構文

```
BEFORE [BREAK] [PROCESSING]
statement ...
END-BEFORE
```

レポーティングモード構文

```
[BEFORE [BREAK] [PROCESSING]
{       statement           }
{       DO statement ... DOEND }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント： **ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

BEFORE BREAK PROCESSING ステートメントは、次の動作の前に処理を実行するために、自動ブレイク処理と組み合わせて使用します。

- ブレイクのコントロールフィールドの値がチェックされる前
- AT BREAK ステートメントで指定されたステートメントが実行される前
- Natural システム関数が評価される前

このステートメントは、ブレイク処理に使用するユーザー定義変数の値を初期化したり、計算を行ったりするためによく使用します（AT BREAK ステートメントを参照）。

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

『*プログラミングガイド*』の次のセクションも参照してください。

- *コントロールブレイク*
- *BEFORE BREAK PROCESSING ステートメント*

- *BEFORE BREAK PROCESSING ステートメントの例*

制限事項

- BEFORE BREAK PROCESSING ステートメントは、次のステートメントのいずれかで始まる処理ループにだけ使用できます。
  - FIND
  - READ
  - HISTOGRAM
  - SORT
  - READ WORK FILE

  BEFORE BREAK PROCESSING ステートメントは、処理ループ内であればどこでも指定することができます。常にこのステートメントを含む処理ループに関連付けられます。 BEFORE BREAK PROCESSING ステートメントは、1 つの処理ループ内で 1 回だけ指定できます。
- BEFORE BREAK PROCESSING ステートメントを、PERFORM BREAK PROCESSING ステートメントと一緒に使用することはできません。

構文説明

statement...	ステートメントの例については、下記の「例」を参照してください。 ブレイク処理が行われない（処理ループに AT BREAK ステートメントが指定されていない）場合、BEFORE BREAK PROCESSING ステートメントで指定したステートメントは*実行されません*。
END-BEFORE	ストラクチャードモードの場合： Natural 予約語 END-BEFORE を使用して、BEFORE BREAK PROCESSING ステートメントを終了させる必要があります。

例

```
** Example 'BBPEX1': BEFORE BREAK PROCESSING
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 SALARY (1)
  2 BONUS  (1,1)
*
1 #INCOME  (P11)
END-DEFINE
*
LIMIT 7
READ EMPLOY-VIEW BY CITY = 'L'
  /*
 BEFORE BREAK PROCESSING
    COMPUTE #INCOME = SALARY (1) + BONUS (1,1)
  END-BEFORE
  /*
  AT BREAK OF CITY
    WRITE NOTITLE  'AVERAGE INCOME FOR' OLD (CITY) 20X AVER(#INCOME) /
  END-BREAK
  /*
  DISPLAY CITY 'NAME' NAME 'SALARY' SALARY (1) 'BONUS' BONUS (1,1)
END-READ
END
```

プログラム **BBPEX1** の出力：

```
        CITY                 NAME           SALARY     BONUS
-------------------- -------------------- ---------- ----------

LA BASSEE            HULOT                   165000      70000
AVERAGE INCOME FOR LA BASSEE                                      235000

LA CHAPELLE ST LUC   GUILLARD                124100      23000
LA CHAPELLE ST LUC   BERGE                   198500      50000
LA CHAPELLE ST LUC   POLETTE                 124090      23000
LA CHAPELLE ST LUC   DELAUNEY                115000      23000
LA CHAPELLE ST LUC   SCHECK                  125600      23000
LA CHAPELLE ST LUC   KREEBS                  184550      50000
AVERAGE INCOME FOR LA CHAPELLE ST LUC                             177306
```

15 CALL

```
CALL [INTERFACE4] operand1 [USING] [operand2] ... 128
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CALL FILE** | **CALL LOOP** | **CALLNAT** | **DEFINE SUBROUTINE** | **ESCAPE** | **FETCH** | **PERFORM**

関連機能グループ：「*プログラムおよびルーチンの呼び出し*」

機能

CALL ステートメントは、Natural プログラムから他の標準プログラミング言語で書かれた外部プログラムまたは関数を呼び出し、CALL の次のステートメントに戻します。

呼び出される側のプログラムまたは関数は、標準 CALL インターフェイスをサポートするプログラミング言語で書くことができます。 1 つ以上の外部プログラムまたは関数を呼び出すために、複数の CALL ステートメントを指定できます。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												
operand1	C	S				A												
operand2	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	

オペランド	ステートメント参照	ダイナミック定義
operand1	可	不可
operand2	可	可

構文要素の説明：

INTERFACE4	オプションのキーワード INTERFACE4 は、外部プログラムのコールに使用するインターフェイスのタイプを指定します。 下記の ***INTERFACE4*** を参照してください。
operand1	呼び出される関数の名前： 呼び出される関数の名前（*operand1*）は、定数または変数で指定できます。プログラムロジックによって呼び出す関数が異なる場合は、長さ1～8の英数字の変数として指定します。変数では、関数名を左詰めで指定する必要があります。
[USING] *operand2*	パラメータ： CALL ステートメントでは、パラメータ（*operand2*）を 128 個まで指定できます。 指定する各パラメータフィールドごとに、1 つのアドレスがパラメータリストで渡されます。

	グループ名を使用した場合、そのグループは個々のフィールドに変換されます。つまり、ユーザーがグループの開始アドレスを指定したい場合、グループの最初のフィールドを指定する必要があります。 **注意:** アプリケーション独立変数（AIV）またはコンテキスト変数をパラメータとしてユーザー出口に渡す場合、そのユーザー出口が新しい AIV またはコンテキスト変数を作成する Natural サブプログラムを呼び出すと、そのパラメータはサブプログラムから戻った後で不正になります。 これは、新しい AIV またはコンテキスト変数がサブプログラム自体、またはサブプログラムから直接または間接的に呼び出された別のオブジェクトで作成されるかどうかに関わらず当てはまります。

リターンコード

呼び出された関数のコンディションコードは、Natural システム関数 RET（リターンコード関数）で取得できます。

例：

```
...
RESET #RETURN(B4)
CALL 'PROG1'
IF RET ('PROG1') > #RETURN
  WRITE 'ERROR OCCURRED IN PROGRAM1'
END-IF
...
```

ユーザー出口

CALL ステートメントで呼び出される外部関数にアクセスできるようにするには、ユーザー出口が必要です。 ユーザー出口は、DLL（ダイナミックリンクライブラリ）内に配置する必要があります。 ユーザー出口の詳細については、次のファイルを参照してください。

%NATDIR%\%NATVERS%\natural\samples\sysexuex\readme.txt

readme.txt ファイルを参照するには、Natural のインストール時にサンプル機能を選択する必要があります。

INTERFACE4

INTERFACE4 キーワードでは、外部プログラムのコールに使用するインターフェイスのタイプを指定します。 このキーワードはオプションです。 このキーワードを指定する場合は、インターフェイスを "Interface4" として定義し、外部プログラムのコールに使用します。

次の表に、INTERFACE4 を使用した場合と、INTERFACE4 使用しなかった場合の CALL ステートメントの違いを示します。

	キーワード INTERFACE4 を使用しない CALL ステートメント	キーワード INTERFACE4 を使用する CALL ステートメント
使用可能なパラメータ数	128	32767
1 つのパラメータの最大データサイズ	64 K	1 GB
配列情報の取得	無	有
ラージオペランドとダイナミックオペランドのサポート	無	有
API 経由でのパラメータアクセス	無	有

以下では次のトピックについて説明します。

- INTERFACE4 - 外部 3GL プログラムインターフェイス
- INTERFACE4 のオペランド構造体
- INTERFACE4 - パラメータアクセス
- エクスポート機能

INTERFACE4 - 外部 3GL プログラムインターフェイス

外部 3GL プログラムのインターフェイスは、Natural の CALL ステートメントで INTERFACE4 を指定するときに、次のように定義します。

```
NATFCT functionname (numparm, parmhandle, traditional)
```

USR_WORD	numparm;	転送するオペランド（*operand2*）の総数が格納される 16 ビット unsigned short 型整数。
void	*parmhandle;	構造体を渡すパラメータへのポインタ。
void	*traditional;	インターフェイスタイプのチェックに使用します（NULL ポインタでない場合は通常の CALL インターフェイス）。

INTERFACE4 のオペランド構造体

INTERFACE4 のオペランド構造体の名前は、「parameter_description」 で次のように定義します。 構造体は、ヘッダーファイル *natuser.h* とともに提供されます。

struct parameter_description			
void *	address	パラメータデータのアドレス。不整列。realloc() および free() は使用できません。	
int	format	フィールドデータフォーマット：NCXR_TYPE_ALPHA など（*natuser.h*）。	
int	length	長さ（小数点以上、該当する場合）。	
int	precision	小数点以下の長さ（該当する場合）。	
int	byte_length	dimensions の整数値のバイト単位のフィールドの長さ（0～IF4_MAX_DIM）。	
int	dimensions	次元の数（0～IF4_MAX_DIM）。	
int	length_all	配列のデータ長の合計（バイト単位）。	
int	flags	ビットごとに OR で結合したフラグビットは次のとおりです。	
		IF4_FLG_PROTECTED：	パラメータは書き込み保護されています。
		IF4_FLG_DYNAMIC：	パラメータはダイナミック変数です。
		IF4_FLG_NOT_CONTIGUOUS：	配列の要素が隣接していません（要素間に空白があります）。
		IF4_FLG_AIV：	このパラメータはアプリケーション独立変数です。
		IF4_FLG_DYNVAR：	パラメータはダイナミック変数です。
		IF4_FLG_XARRAY：	パラメータは x-array です。
		IF4_FLG_LBVAR_0：	次元 0 の下限は可変です。
		IF4_FLG_UBVAR_0：	次元 0 の上限は可変です。
		IF4_FLG_LBVAR_1：	次元 1 の下限は可変です。
		IF4_FLG_UBVAR_1：	次元 1 の上限は可変です。
		IF4_FLG_LBVAR_2：	次元 2 の下限は可変です。
		IF4_FLG_UBVAR_2：	次元 2 の上限は可変です。
int	occurrences[IF4_MAX_DIM]	各次元における配列のオカレンス。	
int	indexfactors[IF4_MAX_DIM]	各次元の配列インデックスファクタ。	
void *	dynp	内部使用に予約済み。	
void *	pops	内部使用に予約済み。	

アドレス要素は、ダイナミック変数の配列および x-array に対してヌルです。 この場合、配列データは全体としてアクセスすることはできませんが、下記のパラメータアクセス機能でアクセスする必要があります。

固定長の変数の固定範囲の配列に対して、配列の内容はアドレス要素を使用してダイレクトにアクセスすることができます。この場合、配列要素（i,j,k）のアドレスは以下のように計算されます（特に配列要素が連続していない場合）。

```
elementaddress = address + i * indexfactors[0] + j * indexfactors[1] + k *
indexfactors[2]
```

配列の次元数が 3 次元よりも小さい場合は、最後の項が除外されます。

INTERFACE4 - パラメータアクセス

パラメータのアクセスに使用する機能の集合があります。 処理フローは次のとおりです。

- 3GL プログラムは `INTERFACE4` オプション付きの `CALL` ステートメントで呼び出され、パラメータが 3GL プログラムに渡されます（上記を参照）。
- その後、3GL プログラムではパラメータデータ自体またはフォーマット、長さ、配列情報などのパラメータに関する情報を検索するために、Naturalのエクスポート機能を使用することができます。
- エクスポート機能はパラメータデータを戻すことにも使用できます。

3GL プログラムから任意のサブプログラムをコールすために、新規パラメータセットを作成し、初期化する機能もあります。 この機能を使用して、3GL プログラムがメモリを上書きしないようにするために、パラメータのアクセスが保証されます （Natural データは安全で、メモリは 3GL プログラムのデータ内を上書きすることはまだ可能です）。

エクスポート機能

以下では次のトピックについて説明します。

- **パラメータ情報の取得**
- **パラメータデータの取得**
- **オペランドデータの書き戻し**
- **パラメータセットの作成、初期化、削除**
- **パラメータセットの作成**
- **パラメータセットの削除**
- **スタティックデータタイプのスカラの初期化**
- **スタティックデータタイプの配列の初期化**
- **ダイナミックデータタイプのスカラの初期化**
- **ダイナミックデータタイプの配列の初期化**

- X-array パラメータのサイズ変更

パラメータ情報の取得

この機能は、任意のパラメータから必要な情報をすべて受け取るために、3GL プログラムで使用します。 情報は、`struct parameter_description`（上記を参照）に返されます。

プロトタイプ：

```
int ncxr_get_parm_info ( int parmnum, void *parmhandle, struct parameter_description
*descr );
```

パラメータの説明：

parmnum	パラメータの順序を表す番号。 これにより、渡されたパラメータリストのパラメータが識別されます。 範囲：0 ～ numparm-1。	
parmhandle	内部パラメータ構造へのポインタ	
descr	struct parameter_description のアドレス	
return	戻り値：	情報：
	0	OK
	-1	無効なパラメータ番号。
	-2	内部エラーが発生しました。
	-7	インターフェイスバージョンの競合。

パラメータデータの取得

この機能は、任意のパラメータからデータを取得するために、3GL プログラムで使用します。

Naturalは、指定したパラメータ番号でパラメータを識別し、指定したバッファサイズのアドレスにパラメータデータを書き出します。

パラメータデータが指定したバッファサイズより長い場合、Naturalは指定した長さにそのデータを切り捨てます。 外部 3GL プログラムは `ncxr_get_parm_info` 関数を、パラメータデータの長さを要求するために使用することができます。

パラメータデータを取得する関数は2つあり、`ncxr_get_parm`はパラメータ全体を取得し（パラメータが配列の場合でも）、`ncxr_get_parm_array` 関数は特定の配列要素を取得します。

（ダイナミックまたはスタティックな）3GL プログラムにより、指定したサイズのメモリが「バッファ」に対して割り当てられない場合、予期できない結果になることがあります。Natural では NULL ポインタのチェックだけが実行されます。

タイプ I2／I4／F4／F8 の変数で、データが切り取られる場合（バッファ長がパラメータの合計長と等しくない）、その結果はマシンのタイプにより異なります（リトルエンディアン／ビッグ

エンディアン）。 アプリケーションによっては、反復を可能にするために、スタティックデータを使用しないようにユーザー出口をプログラムする必要があります。

プロトタイプ：

```
int ncxr_get_parm( int parmnum, void *parmhandle, int buffer_length, void *buffer
) int ncxr_get_parm_array( int parmnum, void *parmhandle, int buffer_length, void
*buffer, int *indexes )
```

This function is identical to `ncxr_get_parm`, except that the indexes for each dimension can be specified. 未使用次元のインデックスは 0 として指定しておく必要があります。

パラメータの説明：

parmnum	パラメータの順序を表す番号。 これにより、渡されたパラメータリストのパラメータが識別されます。 範囲：0 ～ numparm-1。	
parmhandle	内部パラメータ構造へのポインタ	
buffer_length	要求したデータを書き出す必要のあるバッファの長さ。	
buffer	要求したデータを書き出す必要のあるバッファのアドレス。 このバッファは I2/I4/F4/F8 変数にアクセスしやすいように配置しておく必要があります。	
indexes	インデックス情報をもつ配列	
return	戻り値：	情報：
	< 0	情報検索時のエラーを示します。
	-1	無効なパラメータ番号。
	-2	内部エラーが発生しました。
	-3	データは切り捨てられました。
	-4	データが配列ではありません。
	-7	インターフェイスバージョンの競合。
	-100	次元 0 のインデックスが範囲外です。
	-101	次元 1 のインデックスが範囲外です。
	-102	次元 2 のインデックスが範囲外です。
	0	操作に成功しました。
	> 0	正常に行われましたが、（バッファの方がデータよりも長かったため）データの長さはこのバイト数になりました。

オペランドデータの書き戻し

この機能は、任意のパラメータへデータを戻すために、3GLプログラムで使用します。Naturalでは、指定したパラメータ番号でパラメータを識別し、指定したバッファサイズとアドレスからパラメータデータを書き出します。 パラメータデータが指定したバッファサイズより小さい場合、データはパラメータデータの長さに切り捨てられます。つまり、バッファの残りは無視されます。パラメータデータが指定したバッファサイズより大きい場合、データは指定したバッファ長だけコピーされます。パラメータの残りはそのままです。 これは、配列にも同様に当てはまります。パラメータとしてのダイナミック変数については、パラメータのサイズが指定したバッファ長に変更されます。

タイプ I2／I4／F4／F8 の変数で、データが切り取られる場合（バッファ長がパラメータの合計長と等しくない）、その結果はマシンのタイプにより異なります（リトルエンディアン／ビッグエンディアン）。 アプリケーションによっては、反復を可能にするために、スタティックデータを使用しないようにユーザー出口をプログラムする必要があります。

プロトタイプ：

```
int ncxr_put_parm ( int parmnum, void *parmhandle, int buffer_length, void *buffer
); int ncxr_put_parm_array ( int parmnum, void *parmhandle, int buffer_length, void
*buffer, int *indexes );
```

パラメータの説明：

parmnum	パラメータの順序を表す番号。 これにより、渡されたパラメータリストのパラメータが識別されます。 範囲：0 ～ numparm-1。	
parmhandle	内部パラメータ構造体を指すポインタ。	
buffer_length	データを送るバッファのバッファアドレスを元に複写するためのデータの長さ。	
indexes	インデックス情報。	
return	戻り値：	情報：
	< 0	情報複写時のエラーを示します。
	-1	無効なパラメータ番号。
	-2	内部エラーが発生しました。
	-3	指定したデータが多すぎます。 パラメータの長さで複写されました。
	-4	パラメータが配列ではありません。
	-5	パラメータは保護されています（定数または AD=O）。
	-6	「メモリ不足」により、ダイナミック変数のサイズを変更できませんでした。
	-7	インターフェイスバージョンの競合。
	-13	指定したバッファに、不完全な Unicode 文字が含まれています。

	-100	次元 0 のインデックスが範囲外です。
	-101	次元 1 のインデックスが範囲外です。
	-102	次元 2 のインデックスが範囲外です。
	0	操作に成功しました。
	> 0	正常に行われましたが、パラメータ長が指定された長さを超えていたため、パラメータの長さはこのバイト数になりました。

パラメータセットの作成、初期化、削除

3GL プログラムが Natural サブプログラムを呼び出すには、サブプログラムが予期しているパラメータに対応するパラメータセットを構築する必要があります。関数 ncxr_create_parm を使用して、ncxr_if_callnat へのコールで渡すパラメータセットを作成します。 作成されたパラメータセットは、CALL INTERFACE4 ステートメントで 3GL プログラムに渡されるパラメータセットのように、opaque パラメータハンドルによって表されます。したがって、上記のように、新しく作成したパラメータセットは関数 ncxr_put_parm* および ncxr_get_parm* で操作することができます。

新しく作成したパラメータセットは、関数 ncxr_create_parm を呼び出した後は、まだ初期化されていません。 個々のパラメータは下記の ncxr_parm_init* 関数のセットによって特定データタイプに初期化されます。 その後、関数 ncxr_put_parm* および ncxr_get_parm* は個々のパラメータの内容にアクセスするために使用されます。 呼び出す側がパラメータセットを終了した後、それらはパラメータハンドルを削除する必要があります。 したがって、ncxr_if4_callnat を通して呼び出されるサブプログラムのパラメータセットの作成と使用の一般的なシーケンスは、次のようになります。

```
ncxr_create_parm
ncxr_init_ parm*
ncxr_init_ parm*
...
ncxr_put_ parm*
ncxr_put_ parm*
...
ncxr_get_parm_info*
ncxr_get_parm_info*
...
ncxr_if4_callnat
...
ncxr_get_parm_info*
ncxr_get_parm_info*
...
ncxr_get_ parm*
ncxr_get_ parm*
```

```
...
ncxr_delete_parm
```

パラメータセットの作成

関数 ncxr_create_parm を使用して、ncxr_if_callnat へのコールで渡すパラメータセットを作成します。

プロトタイプ：

```
int ncxr_create_parm( int parmnum, void** pparmhandle )
```

パラメータの説明：

parmnum	作成するパラメータの数。	
pparmhandle	作成されたパラメータハンドルへのポインタ。	
return	戻り値：	情報：
	< 0	エラー：
	-1	パラメータカウントが不正です。
	-2	内部エラーが発生しました。
	-6	メモリ不足の状態です。
	0	操作に成功しました。

パラメータセットの削除

ncxr_create_parm で作成されたパラメータセットを削除するには、関数 ncxr_delete_parm を使用します。

プロトタイプ：

```
int ncxr_delete_parm( void* parmhandle )
```

パラメータの説明：

parmhandle	削除するパラメータハンドルへのポインタ。	
return	戻り値：	情報：
	< 0	エラー：
	-2	内部エラーが発生しました。
	0	操作に成功しました。

スタティックデータタイプのスカラの初期化

プロトタイプ：

```
int ncxr_init_parm_s( int parmnum, void *parmhandle, char format, int length, int
precision, int flags );
```

パラメータの説明：

parmnum	パラメータの順序を表す番号。 これにより、渡されたパラメータリストのパラメータが識別されます。 範囲：0 ～ numparm-1。	
parmhandle	パラメータハンドルへのポインタ。	
format	パラメータのフォーマット。	
length	パラメータの長さ。	
precision	パラメータの精度。	
flags	フラグの組み合わせ。 IF4_FLG_PROTECTED	
return	戻り値：	情報：
	< 0	エラー：
	-1	無効なパラメータ番号。
	-2	内部エラーが発生しました。
	-6	メモリ不足の状態です。
	-8	フォーマットが正しくありません。
	-9	無効な長さまたは精度。
	0	操作に成功しました。

スタティックデータタイプの配列の初期化

プロトタイプ：

```
int ncxr_init_parm_sa( int parmnum, void *parmhandle, char format, int length, int
precision, int dim, int *occ, int flags );
```

パラメータの説明：

parmnum	パラメータの順序を表す番号。 これにより、渡されたパラメータリストのパラメータが識別されます。 範囲：0 ～ numparm-1。	
parmhandle	パラメータハンドルへのポインタ。	
format	パラメータのフォーマット。	
length	パラメータの長さ。	
precision	パラメータの精度。	
dim	配列の次元。	
occ	次元ごとのオカレンス数。	
flags	フラグの組み合わせ。 IF4_FLG_PROTECTED IF4_FLG_LBVAR_0 IF4_FLG_UBVAR_0 IF4_FLG_LBVAR_1 IF4_FLG_UBVAR_1 IF4_FLG_LBVAR_2 IF4_FLG_UBVAR_2	
return	戻り値：	情報：
	< 0	エラー：
	-1	無効なパラメータ番号。
	-2	内部エラーが発生しました。
	-6	メモリ不足の状態です。
	-8	フォーマットが正しくありません。
	-9	無効な長さまたは精度。
	-10	無効な次元数。
	-11	変数制限の無効な組み合わせ。
	0	操作に成功しました。

ダイナミックデータタイプのスカラの初期化

プロトタイプ：

```
int ncxr_init_parm_d( int parmnum, void *parmhandle, char format, int flags );
```

パラメータの説明：

parmnum	パラメータの順序を表す番号。 これにより、渡されたパラメータリストのパラメータが識別されます。 範囲：0 ～ numparm-1。	
parmhandle	パラメータハンドルへのポインタ。	
format	パラメータのフォーマット。	
flags	フラグの組み合わせ。 IF4_FLG_PROTECTED	
return	戻り値：	情報：
	< 0	エラー：
	-1	無効なパラメータ番号。
	-2	内部エラーが発生しました。
	-6	メモリ不足の状態です。
	-8	フォーマットが正しくありません。
	0	操作に成功しました。

ダイナミックデータタイプの配列の初期化

プロトタイプ：

```
int ncxr_init_parm_da( int parmnum, void *parmhandle, char format, int dim, int
*occ, int flags );
```

パラメータの説明：

parmnum	パラメータの順序を表す番号。 これにより、渡されたパラメータリストのパラメータが識別されます。 範囲：0 ～ numparm-1。	
parmhandle	パラメータハンドルへのポインタ。	
format	パラメータのフォーマット。	
dim	配列の次元。	
occ	次元ごとのオカレンス数。	
flags	フラグの組み合わせ。 IF4_FLG_PROTECTED IF4_FLG_LBVAR_0 IF4_FLG_UBVAR_0 IF4_FLG_LBVAR_1 IF4_FLG_UBVAR_1 IF4_FLG_LBVAR_2 IF4_FLG_UBVAR_2	
return	戻り値：	情報：
	< 0	エラー：

	-1	無効なパラメータ番号。
	-2	内部エラーが発生しました。
	-6	メモリ不足の状態です。
	-8	フォーマットが正しくありません。
	-10	無効な次元数。
	-11	変数制限の無効な組み合わせ。
	0	操作に成功しました。

X-array パラメータのサイズ変更

プロトタイプ：

```
int ncxr_resize_parm_array( int parmnum, void *parmhandle, int *occ );
```

パラメータの説明：

parmnum	パラメータの順序を表す番号。 これにより、渡されたパラメータリストのパラメータが識別されます。 範囲：0 ～ numparm-1。	
parmhandle	パラメータハンドルへのポインタ。	
occ	次元当たりの新規オカレンス数。	
return	戻り値：	情報：
	< 0	エラー：
	-1	無効なパラメータ番号。
	-2	内部エラーが発生しました。
	-6	メモリ不足の状態です。
	-12	オペランドが（指定された次元の 1 つで）サイズ変更可能ではありません。
	0	操作に成功しました。

すべての関数プロトタイプがファイル *natuser.h* で宣言されます。

16 CALL FILE

ストラクチャードモード構文

```
CALL FILE 'program-name' operand1 operand2
statement ...
END-FILE
```

レポーティングモード構文

```
CALL FILE 'program-name' operand1 operand2
statement ...
[LOOP]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CALL** | **CALL LOOP** | **CALLNAT** | **DEFINE SUBROUTINE** | **ESCAPE** | **FETCH** | **PERFORM**

関連機能グループ：「*プログラムおよびルーチンの呼び出し*」

機能

CALL FILE ステートメントは、Adabas 以外のファイルからレコードを読み取って Natural プログラムに返す、Natural 以外のプログラムを呼び出すために使用します。

制限事項

ステートメント AT BREAK、AT START OF DATA、および AT END OF DATA を、CALL FILE 処理ループ内で使用することはできません。

構文説明

オペランド定義テーブル：

オペランド	構文要素			フォーマット	ステートメント参照	ダイナミック定義
operand1		S	A	A U N P I F B D T L C	可	可
operand2		S	A	G A U N P I F B D T L C	可	可

構文要素の説明：

'*program-name*'	呼び出される Natural 以外のプログラムの名前。
operand1	コントロールフィールド：*operand1* は制御情報の提供に使用されます。
operand2	*operand2* では、レコードエリアを定義します。 読み取るレコードのフォーマットは、レコード内の最初のフィールド名に続けてフィールド定義（または FILLER *n*X）エントリを使用して定義します。レコードフォーマットを定義するために使用されるフィールドは、Natural プログラム内で事前に定義しないでください。これは Natural がフィールドを連続ストレージに割り付けることを保証するためです。
statement ...	CALL FILE ステートメントは、処理ループを開始します。この処理ループは、ESCAPE または STOP ステートメントで終了する必要があります。さまざまな条件に対応して CALL FILE ループから抜け出せるように、複数の ESCAPE ステートメントを指定することも可能です。
END-FILE	END-FILE ステートメントを使用して、処理ループを閉じる必要があります。

例

呼び出す側のプログラム：

```
** Example 'CFIEX1': CALL FILE
************************************************************************
DEFINE DATA LOCAL
1 #CONTROL (A3)
1 #RECORD
  2 #A     (A10)
  2 #B     (N3.2)
  2 #FILL1 (A3)
  2 #C     (P3.1)
END-DEFINE
```

```
*
CALL FILE 'USER1' #CONTROL #RECORD
  IF #CONTROL = 'END'
    ESCAPE BOTTOM
  END-IF
END-FILE
/*****************************
/* ... PROCESS RECORD ...
/*****************************
END
```

上記の例で呼び出されたプログラムが、Naturalプログラムに渡すレコードのバイトレイアウトは、次のとおりです。

```
CONTROL #A       #B   FILLER  #C
(A3)   (A10)   (N3.2)  3X   (P3.1)

xxx xxxxxxxxxx xxxxx   xxx   xxx
```

呼び出される側の **COBOL** プログラム：

```
ID DIVISION.
PROGRAM-ID. USER1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT USRFILE ASSIGN UT-S-FILEUSR.
DATA DIVISION.
FILE SECTION.
FD   USRFILE RECORDING F LABEL RECORD OMITTED
     DATA RECORD DATA-IN.
01   DATA-IN         PIC X(80).
LINKAGE SECTION.
01   CONTROL-FIELD   PIC XXX.
01   RECORD-IN       PIC X(21).
PROCEDURE DIVISION USING CONTROL-FIELD RECORD-IN.
BEGIN.
     GO TO FILE-OPEN.
FILE-OPEN.
     OPEN INPUT USRFILE
     MOVE SPACES TO CONTROL-FIELD.
     ALTER BEGIN TO PROCEED TO FILE-READ.
FILE-READ.
     READ USRFILE INTO RECORD-IN
          AT END
          MOVE 'END' TO CONTROL-FIELD
          CLOSE USRFILE
          ALTER BEGIN TO PROCEED TO FILE-OPEN.
     GOBACK.
```

17 CALL LOOP

ストラクチャードモード構文

```
CALL LOOP operand1 [operand2] ...40
statement ...
END-LOOP
```

レポーティングモード構文

```
CALL LOOP operand1 [operand2] ...40
statement ...
[LOOP]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CALL** | **CALL FILE** | **CALLNAT** | **DEFINE SUBROUTINE** | **ESCAPE** | **FETCH** | **PERFORM**

関連機能グループ：「*プログラムおよびルーチンの呼び出し*」

機能

CALL LOOP ステートメントは、Natural 以外のプログラムを呼び出す処理ループを生成します。

CALL ステートメントとは異なり、CALL LOOP ステートメントは、処理ループ内で繰り返し Natural 以外のプログラムを呼び出します。CALL 処理の詳細については、CALL ステートメントを参照してください。

制限事項

ステートメント AT BREAK、AT START OF DATA、および AT END OF DATA を、CALL LOOP 処理ループ内で使用することはできません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A												可	不可
operand2	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C		可	可

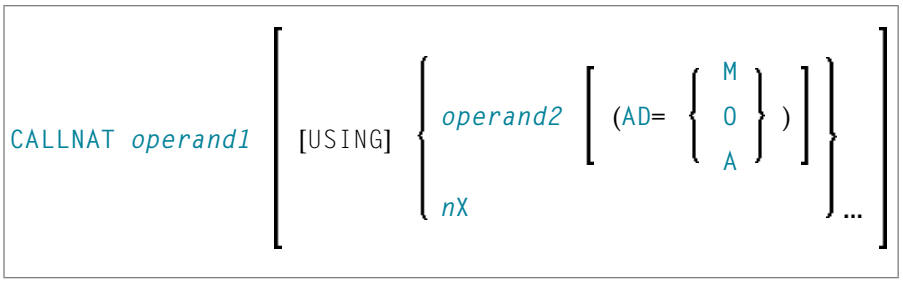
構文要素の説明：

operand1	呼び出される Natural プログラムの名前は、定数または変数で指定できます。プログラムロジックによって呼び出すプログラムが異なる場合は、長さ1～8の英数字の変数として指定します。 変数では、プログラム名を左詰めで指定する必要があります。
operand2	CALL LOOP ステートメントは、最大 40 個までのパラメータを受け渡すことができます。 パラメータリストの構成は、CALL ステートメントと同じです。 パラメータリストのフィールドは、CALL LOOP ステートメント自体で最初に定義するか、または事前に定義しておくことができます。
statement ...	CALL LOOP ステートメントによって開始された処理ループは、ESCAPE ステートメントで終了する必要があります。
END-LOOP	Natural 予約語 END-LOOP を使用して、処理ループを閉じる必要があります。

例

```
DEFINE DATA LOCAL
1 PARAMETER1 (A10)
END-DEFINE
CALL LOOP 'ABC' PARAMETER1
  IF PARAMETER1 = 'END'
     ESCAPE BOTTOM
  END-IF
END-LOOP
END
```

18 CALLNAT



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CALL** | **CALL FILE** | **CALL LOOP** | **DEFINE SUBROUTINE** | **ESCAPE** | **FETCH** | **PERFORM**

関連機能グループ：「*プログラムおよびルーチンの呼び出し*」

機能

CALLNAT ステートメントは、Natural サブプログラムを実行するために呼び出します。 Natural サブプログラムは、CALLNAT ステートメントによってのみ呼び出すことができます。サブプログラムを単独で実行することはできません。

CALLNAT ステートメントを実行すると、呼び出し元オブジェクト（CALLNAT ステートメントを含んでいるオブジェクト）は中断され、呼び出したサブプログラムが実行されます。 サブプログラムの実行は、END ステートメントに到達するまで、または ESCAPE ROUTINE ステートメントの実行によってサブプログラムの処理が停止されるまで続きます。 どちらの場合でも、呼び出し元オブジェクトのその後の処理は、CALLNAT ステートメントの次にあるステートメントから継続されます。

Notes:

1. サブプログラムは、他のサブプログラムを交互に呼び出すことができます。
2. サブプログラムには、呼び出し側オブジェクトが使用するグローバルデータエリアへのアクセスがありません。 サブプログラムからサブルーチンやヘルプルーチンを呼び出す場合は、サブプログラムで独自のグローバルデータエリアを作成して、サブルーチンまたはヘルプルーチンと共有することができます。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1	C	S				A													可	不可
operand2	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	O	可	可

構文要素の説明：

operand1	サブプログラム名： *operand1* として、呼び出すサブプログラムの名前を指定します。 名前は、1～8 文字の定数または変数で指定できます。プログラムロジックによって呼び出すサブプログラムが異なる場合は、長さ 1～8 の英数字の変数として指定します。 サブプログラム名に、アンパサンド（&）を含めることができます。これは、実行時にシステム変数 *LANGUAGE の現在の値で置き換えられます。 これにより、入力時の言語に応じて、入力処理ごとに異なるサブプログラムを呼び出すことができます。
operand2	パラメータ： パラメータをサブプログラムに渡す場合、パラメータリストの構成は、DEFINE DATA PARAMETER ステートメントで定義する必要があります。 CALLNAT ステートメントで指定したパラメータだけが、呼び出し元オブジェクトからサブプログラムに有効なデータです。 デフォルトでは、パラメータは*参照*によって渡されます。つまり、データはアドレスパラメータを介して転送され、パラメータ値自体は移動されません。 ただし、パラメータを*値*によって渡す、つまり実際のパラメータ値を渡すこともできます。 これを行うには、サブプログラムの DEFINE DATA PARAMETER ステートメントで、これらのフィールドを BY VALUE または BY VALUE RESULT オプション付きで定義します（DEFINE DATA ステートメントの説明にある *parameter-data-definition* を参照）。 ■ パラメータを*参照*によって渡す場合は、次のことが適用されます。呼び出し元オブジェクトのパラメータの順番、フォーマット、および長さは、呼び出されるサブプログラムの DEFINE DATA PARAMETER 構造の順番、フォーマット、および長さと正確に一致させる必要があります。 呼び出し元オブジェクトの変数名と呼び出されるサブプログラムの変数名は、異なっていてもかまいません。 ■ パラメータを*値*によってで渡す場合は、次のことが適用されます。呼び出し元オブジェクトのパラメータの順番は、呼び出されるサブプログラムの DEFINE DATA PARAMETER 構造の順番と正確に一致させる必要があります。 呼び出し元オブジェクトの変数のフォーマットと長さは、サブプログラムの変数と異なっていてもかまいません。ただし、データ転送の互換性が必要です（詳細については、『*プログラミングガイド*』の「*演算割り当てのルール*」、「*データ転送*」にある対応表を参照）。 呼び出し元オブジェクトと呼び出されるサブプログラムで異なる変数名を使用できます。 サブプログラムで変更されたパラメータの値を呼び出し元オブジェクトに戻す場合、これらのフィールドを BY VALUE RESULT 付き

	で定義する必要があります。 BY VALUE では（RESULT を使用せずに）、修正されたパラメータ値を呼び出し側オブジェクトに戻すことはできません（AD指定とは関係ありません。以下も参照してください）。 **注意:** BY VALUE では、パラメータ値の内部コピーが作成されます。 サブプログラムがこのコピーにアクセスして、その値を変更しても、呼び出し元オブジェクトのオリジナルのパラメータ値には影響しません。 BY VALUE RESULT では、同様に内部的なコピーが作成されます。ただし、サブプログラム終了後、オリジナルのパラメータ値はコピーの値（変更された）で上書きされます。 いずれのパラメータ引き渡し方法にも、次が適用されます。 *operand2*としてグループを指定すると、グループ内にある個々のフィールドがサブプログラムに渡されます。つまり、これらの各フィールドに対応するフィールドを、サブプログラムのパラメータデータエリアに定義する必要があります。 呼び出されたサブプログラムのパラメータデータエリアでは、グループの再定義はREDEFINEブロック内だけで行うことができます。 配列を渡す場合、サブプログラムのパラメータデータエリアの次元数とオカレンス数はCALLNATパラメータリストと同じにする必要があります。 **注意:** インデックス付きグループの一部として定義された配列の複数オカレンスをCALLNATステートメントで渡す場合は、サブプログラムのパラメータデータエリア内の対応するフィールドを再定義することはできません。再定義すると、不正なアドレスが渡されます。 COMPOPT コマンドのオプション PCHECK を ON に設定すると、CALLNAT ステートメントで指定されたパラメータの数値、フォーマット、長さ、および配列インデックスの限度がコンパイラによってチェックされます。 また、DEFINE DATA PARAMETER ステートメントの OPTIONAL 機能もパラメータチェックで考慮されます。
AD=	属性定義： *operand2* が変数の場合は、次のいずれかの方法でマークすることができます。

AD=O	変更不可。セッションパラメータ AD=O を参照してください。 **注意:** 内部的には、AD=O は BY VALUE と同様に処理されます（DEFINE DATA ステートメントの説明にある *parameter-data-definition*を参照）。
AD=M	変更可。セッションパラメータ AD=M を参照してください。 これはデフォルト設定です。
AD=A	入力のみ。セッションパラメータ AD=A を参照してください。
operand2 が定数の場合は、AD を明示的に指定することはできません。 定数には常に AD=O が適用されます。	

***n*X**	表記 *n*X を使用して、次の *n* 個のパラメータをスキップするように指定することができます（例えば、1X を使用すると次のパラメータがスキップされ、3X を使用すると次の3つのパラメータがスキップされます）。これは、次の *n* 個のパラメータでは、値がサブプログラムに渡されないことを意味します。 *n* の値の有効範囲は 1～4096 です。 スキップするパラメータは、サブプログラムの DEFINE DATA PARAMETER ステートメント内のキーワード OPTIONAL を使用して定義する必要があります。 OPTIONAL は、値を呼び出し側オブジェクトからこのようなパラメータに渡すこともできるということを意味します。

ダイナミック変数を使用したパラメータ引き渡し

ダイナミック変数は、呼び出されたプログラムオブジェクト（CALLNAT、PERFORM）へのパラメータとして渡すことができます。 ダイナミック変数の値スペースは連続しているので、参照渡しが可能です。 値渡しを使用すると、呼び出し元の変数定義がソースオペランドとして割り当てられ、パラメータ定義が応答先オペランドとして割り当てられます。 さらに、値による呼び出しは、結果として反対方向への動作を変更します。 参照による呼び出しを使用する場合、変数定義およびパラメータ定義は DYNAMIC である必要があります。 そのうちの 1 つだけが DYNAMIC の場合、ランタイムエラーが発生します。 値による呼び出し（の結果）の場合は、すべての組み合わせが可能です。

次の表は、パラメータの転送に関して、呼び出し元のスタティックおよびダイナミックに定義された変数と、スタティックおよびダイナミックに定義されたパラメータの有効な組み合わせを示しています。

参照渡し

呼び出し元の *operand2*	パラメータ定義	
	スタティック	ダイナミック
スタティック	○	×
ダイナミック	×	○

ダイナミック変数 A または B のフォーマットは一致している必要があります。

値渡し（結果）

呼び出し元の *operand2*	パラメータ定義	
	スタティック	ダイナミック
スタティック	○	○
ダイナミック	○	○

Note: スタティック／ダイナミックまたはダイナミック／スタティック定義を使用する場合は、割り当ての際にデータ転送の規則によって値が切り捨てられることがあります。

例

例 1

呼び出す側のプログラム：

```
** Example 'CNTEX1': CALLNAT
************************************************************************
DEFINE DATA LOCAL
1 #FIELD1 (N6)
1 #FIELD2 (A20)
1 #FIELD3 (A10)
END-DEFINE
*
CALLNAT 'CNTEX1N' #FIELD1 (AD=M) #FIELD2 (AD=O) #FIELD3 'P4 TEXT'
*
WRITE '=' #FIELD1 '=' #FIELD2 '=' #FIELD3
*
END
```

呼び出される側のサブプログラム **CNTEX1N**：

```
** Example 'CNTEX1N': CALLNAT  (called by CNTEX1)
************************************************************************
DEFINE DATA PARAMETER
1 #FIELDA (N6)
1 #FIELDB (A20)
1 #FIELDC (A10)
1 #FIELDD (A7)
END-DEFINE
*
*
#FIELDA := 4711
*
#FIELDB := 'HALLO'
*
#FIELDC := 'ABC'
*
WRITE '=' #FIELDA '=' #FIELDB '=' #FIELDC '=' #FIELDD
```

```
*
END
```

例 2

呼び出す側のプログラム：

```
** Example 'CNTEX2': CALLNAT
************************************************************************
DEFINE DATA LOCAL
1 #ARRAY1  (N4/1:10,1:10)
1 #NUM     (N2)
END-DEFINE
*
*
CALLNAT 'CNTEX2N' #ARRAY1 (2:5,*)
*
FOR #NUM 1 TO 10
  WRITE #NUM #ARRAY1(#NUM,1:10)
END-FOR
*
END
```

呼び出される側のサブプログラム CNTEX2N：

```
** Example 'CNTEX2N': CALLNAT  (called by CNTEX2)
************************************************************************
DEFINE DATA
PARAMETER
1 #ARRAY (N4/1:4,1:10)
LOCAL
1 I      (I2)
END-DEFINE
*
*
FOR I 1 10
  #ARRAY(1,I) := I
  #ARRAY(2,I) := 100 + I
  #ARRAY(3,I) := 200 + I
  #ARRAY(4,I) := 300 + I
END-FOR
*
END
```

19 CLOSE CONVERSATION

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DEFINE DATA CONTEXT** | **OPEN CONVERSATION**

機能

CLOSE CONVERSATION ステートメントは、Natural RPC と一緒に使用します。 これにより、クライアントは会話を閉じることができます。 現在の会話、開かれた別の会話、または開かれたすべての会話を閉じることができます。

Note: 別のライブラリにログオンしても、会話が自動的に閉じることはありません。

構文説明

オペランド定義テーブル：

オペランド	構文要素			フォーマット	ステートメント参照	ダイナミック定義
operand1		S	A	I	可	不可

構文要素の説明：

operand1	閉じられる会話： 開かれている特定の会話を閉じるには、その ID を *operand1* として指定します。 *operand1* はフォーマット／長さが I4 の変数にする必要があります。
***CONVID**	現在の会話を閉じるには、*CONVID を指定します。 現在の会話の ID は、システム変数 *CONVID の値によって決定されます。
ALL	開かれているすべての会話を閉じる場合は、ALL を指定します。

詳細と例

『*Natural リモートプロシージャコール（RPC）*』ドキュメントの次のセクションを参照してください。

- *会話型モードでの Natural RPC の動作*
- *会話型 RPC の使用*

20 CLOSE DIALOG

```
CLOSE DIALOG [USING] [DIALOG-ID] { operand1    }
                                 { *DIALOG-ID  }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**OPEN DIALOG** | **PROCESS GUI** | **SEND EVENT**

関連機能グループ：「*イベントドリブンプログラミング*」

機能

CLOSE DIALOG ステートメントは、ダイナミックにダイアログを閉じるために使用します。

Note: モーダルダイアログがダイアログ階層で子である場合、モーダルダイアログの親を閉じるとデッドロックが生じるので、親を閉じないでください。

構文説明

オペランド定義テーブル：

オペランド	構文要素	フォーマット	ステートメント参照	ダイナミック定義
operand1	S	I	可	不可

構文要素の説明：

operand1	識別子： *operand1* は、閉じられるダイアログの識別子です。
***DIALOG-ID**	現在のダイアログを閉じるには、ダイアログの現在のインスタンスに割り当てられている ID を保持しているシステム変数 *DIALOG-ID を指定します。

詳細と例

『*プログラミングガイド*』の「イベントドリブンプログラミング手法」を参照してください。

21 CLOSE PRINTER

```
CLOSE PRINTER  { (logical-printer-name) }
               { (printer-number)       }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** |**DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

CLOSE PRINTER ステートメントは、特定のプリンタを閉じます。 このステートメントを使用して、閉じるプリンタをプログラム内で明示的に指定します。

また、次のいずれかの場合は、プリンタが自動的に閉じられます。

- 同じプリンタをもう一度定義する DEFINE PRINTER ステートメントが実行された場合。
- コマンドモードになった場合。

構文説明

logical-printer-name	論理プリンタ名： *logical-printer-name*で、どのプリンタを閉じるかを指定します。 このプリンタ名およびプリンタ番号は、対応する **DEFINE PRINTER** ステートメントで定義したプリンタと同じものです。 *logical-printer-name*の命名規則はユーザー定義変数と同じ規則です。詳細については、『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。
printer-number	プリンタ番号： *logical-printer-name*の代わりに、*printer-number*を定義して閉じるプリンタを指定することもできます。 *printer-number* は、0～31 の範囲にある番号になります。 この番号は、DISPLAY/WRITE または **DEFINE PRINTER** ステートメントでも使用します。 プリンタ番号 0 はハードコピープリンタを示します。

例

```
** Example 'CLPEX1': CLOSE PRINTER
************************************************************************
DEFINE DATA LOCAL
1 EMP-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #I-NAME (A20)
END-DEFINE
*
DEFINE PRINTER (PRT01=1)
*
REPEAT
  INPUT 'SELECT PERSON' #I-NAME
  IF #I-NAME = ' '
    STOP
  END-IF
  FIND EMP-VIEW WITH NAME = #I-NAME
    WRITE (PRT01) 'NAME         :' NAME ',' FIRST-NAME
           /      'PERSONNEL-ID :' PERSONNEL-ID
           /      'BIRTH        :' BIRTH (EM=YYYY-MM-DD)
  END-FIND
  /*
  CLOSE PRINTER (PRT01)
  /*
END-REPEAT
END
```

22 CLOSE WORK FILE

```
CLOSE WORK [FILE] work-file-number
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DEFINE WORK FILE** | **READ WORK FILE** | **WRITE WORK FILE**

関連機能グループ：「*ワークファイル／PC ファイルの制御*」

機能

CLOSE WORK FILE ステートメントは、特定のワークファイルをクローズします。 これにより、クローズするワークファイルをプログラム内で指定できます。

次の場合は、ワークファイルが自動的にクローズされます。

- コマンドモードになった場合。
- READ WORK FILE ステートメントの実行中にエンドオブファイル条件が発生した場合。
- 別のファイルを関連するワークファイル番号に割り当てる DEFINE WORK FILE ステートメントが実行される前。

構文説明

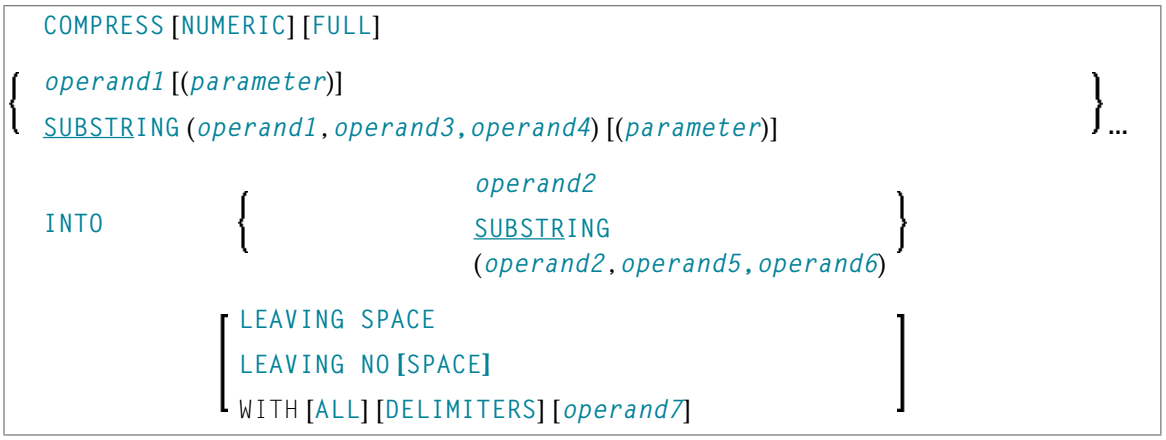
work-file-number	閉じるワークファイルの番号（Natural に定義した）。

例

```
** Example 'CWFEX1': CLOSE WORK FILE
************************************************************************
DEFINE DATA LOCAL
1 W-DAT   (A20)
1 REC-NUM (N3)
1 I       (P3)
END-DEFINE
*
REPEAT
  READ WORK FILE 1 ONCE W-DAT  /* READ MASTER RECORD
  /*
```

```
  AT END OF FILE
    ESCAPE BOTTOM
  END-ENDFILE
  INPUT 'PROCESSING FILE' W-DAT (AD=O)
      / 'ENTER RECORDNUMBER TO DISPLAY' REC-NUM
  IF REC-NUM = 0
    STOP
  END-IF
    FOR I = 1 TO REC-NUM
    /*
    READ WORK FILE 1 ONCE W-DAT
    /*
    AT END OF FILE
      WRITE 'RECORD-NUMBER TOO HIGH, LAST RECORD IS'
      ESCAPE BOTTOM
    END-ENDFILE
  END-FOR
  I := I - 1
  WRITE 'RECORD' I ':' W-DAT
  /*
  CLOSE WORK FILE 1
  /*
END-REPEAT
END
```

23 COMPRESS



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ASSIGN** | **COMPUTE** | **EXAMINE** | **MOVE** | **MOVE ALL** | **SEPARATE**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

COMPRESS ステートメントは、1つ以上のオペランドの内容を単一の英数字フィールドへ転送（結合）するために使用します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												オペランド参照	ダイナミック定義
operand1	C	S	A	G	N	A	U	N	P	I	F	B	D	T			G	O	可	不可
operand2		S				A	U					B							可	可
operand3	C	S						N	P	I		B*							可	不可
operand4	C	S						N	P	I		B*							可	不可
operand5	C	S						N	P	I		B*							可	不可
operand6	C	S						N	P	I		B*							可	不可
operand7	C	S				A	U					B							可	不可

* *operand3* のフォーマット B、*operand4*、*operand5*、および *operand6* は、4 以下の長さでのみ使用できます。

構文要素の説明：

NUMERIC	このオプションは、符号文字や小数文字の扱いを指定します。 NUMERIC を指定しないと、数値の小数点や符号は値が転送される前に除外されます。 次に例を示します。 `COMPRESS -123  1.23 INTO #TARGET WITH DELIMITER '*'` `Content of #TARGET is: 123*123` NUMERIC を指定すると、数値の小数点や符号もターゲットフィールドに転送されます。 浮動小数点の値については、NUMERICが指定されているかどうかに関係なく、小数点と符号が転送されます。 例 **1**： `COMPRESS NUMERIC -123  1.23 INTO #TARGET WITH DELIMITER '*'` `Content of #TARGET is: -123*1.23` 例 **2**： `COMPRESS NUMERIC 'ABC' -0056.00 -0056.10 -0056.01 INTO #TARGET WITH DELIMITER '*'` `Content of #TARGET is: ABC*-56*-56.1*-56.01` 例 **3**： `COMPRESS NUMERIC FULL 'ABC' -0056.00 -0056.10 -0056.01 INTO #TARGET WITH DELIMITER '*'` `Content of #TARGET is: ABC*-0056.00*-0056.10*-0056.01`
FULL	FULLを指定しないと、値が転送される前に次のものがソースフィールドから削除されます。 ■ フォーマット N、P、または I のフィールドの小数点の前にある先頭のゼロ ■ フォーマット N または P のフィールドの小数点の後にある末尾のゼロ ■ フォーマット A のフィールドの末尾にある空白 ■ フォーマット B のフィールドの先頭にあるバイナリの 0

	すべて 0 の数値ソースフィールドについては 1 つの "0" が転送されます。 次に例を示します。 `COMPRESS 'ABC '  001 INTO #TARGET WITH DELIMITER '*'` `Content of #TARGET is: ABC*1` FULL を指定すると、ソースフィールドの実際の長さの値がターゲットフィールドに送られます。 つまり、次のようになります。 ■ フォーマット N、P、または I のフィールドの小数点の前にある先頭のゼロ ■ フォーマット N または P のフィールドの小数点の後にある末尾のゼロ ■ フォーマット A のフィールドの末尾にある空白 ■ フォーマット B のフィールドの先頭にあるバイナリの 0 これらは入力したとおりに表示されます。 次に例を示します。 `COMPRESS FULL 'ABC '  001 INTO #TARGET WITH DELIMITER '*'` `Content of #TARGET is: ABC *001`
operand1	ソースフィールド： *operand1* では、内容が転送されるフィールドを指定します。 **注意:** *operand1* がフォーマット A または B でない場合、その内容は転送される前に英数字表現に変換されます。 必要に応じて、英数字表現は切り捨てられます。 *operand1* が時刻変数（フォーマット T）の場合は、変数内容のうち時刻コンポーネントのみが転送され、日付コンポーネントは転送されません。
operand2	ターゲットフィールド： *operand2* では、ソースフィールドの値を受け取るフィールドを指定します。 ターゲットフィールドがフォーマット U（Unicode）であり、フォーマット B のソースフィールドが含まれている場合、送信するバイナリフィールドの長さを同じにする必要があります。
LEAVING SPACE	追加オプションを一切指定しないで COMPRESS ステートメントを使用したか、または LEAVING SPACE（デフォルトでも適用される）を指定した場合、ターゲットフィールド内の値は互いに空白で区切られます。
LEAVING NO SPACE	LEAVING NO SPACE が指定されると、ターゲットフィールド内の各値の間には、空白も文字も入りません。
parameter	*parameter* では、セッションパラメータ PM またはセッションパラメータ DF を指定できます。

	PM=I	右から左に記述する言語をサポートするために、PM=I を指定して、*operand1* の値を逆（右から左）方向で *operand2* に渡すことができます。例えば、次のステートメントの結果として、#B の内容は "ZYXABC" になります。 `MOVE 'XYZ' TO #A` `COMPRESS #A (PM=I) 'ABC' INTO #B` `LEAVING NO SPACE` *operand1* の末尾にある空白はすべて削除され（FULL を指定した場合を除く）、値は 1 文字ずつ逆向きに並べ換えられて *operand2* に転送されます。
	DF	*operand1* が日付変数の場合、変数に対する *parameter* としてセッションパラメータ DF を指定できます。
SUBSTRING (*operand1*, *operand3*, *operand4*)	*operand1* が英数字（A）、Unicode（U）、またはバイナリ（B）フォーマットの場合、SUBSTRING オプションを使用して、ソースフィールドの特定部分だけを転送できます。フィールド名（*operand1*）の後に、まず開始位置（*operand3*）を指定してから、フィールドの転送される部分の長さ（*operand4*）を指定してください。	
INTO SUBSTRING (*operand2*, *operand5*, *operand6*)	INTO 節に SUBSTRING オプションを使用してソース値をターゲットフィールドの特定部分に転送することもできます。 どちらの場合も、COMPRESS ステートメントの SUBSTRING オプションの使用は、MOVE ステートメントの SUBSTRING オプションに対応しています。SUBSTRING オプションの詳細については、MOVE ステートメントを参照してください。	
WITH DELIMITERS	DELIMITERS オプションは、ターゲットフィールドの値を特定の文字で区切ることができます。 WITH DELIMITERS を *operand7* なしで指定した場合、値は INPUT 区切り文字（セッションパラメータ ID で定義した値）で区切られます。	
WITH DELIMITERS *operand7*	WITH DELIMITERS *operand7* を指定すると、値は *operand7* に指定した文字で区切られます。*operand7* は、1 文字にする必要があります。*operand7* が変数の場合は、そのフォーマット／長さを（A1）または（B1）にする必要があります ターゲットフィールドがフォーマット A または B の場合は、デリミタ文字のフォーマット／長さを（A1）、（B1）、または（U1）にする必要があります。 ターゲットフィールドがフォーマット U（Unicode）の場合は、デリミタ文字のフォーマット／長さを（A1）、（B2）、または（U1）にする必要があります。	

WITH ALL	ALLを指定しないと、ターゲットフィールドでは、実際に転送された値の間にのみデリミタ文字が挿入されます。 次に例を示します。 `COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH DELIMITERS '*'` `Content of #TARGET is: A*C` ALLを指定すると、ターゲットフィールドでは、実際には転送されない空値に対してもデリミタ文字が挿入されます。デリミタ文字の個数は、ソースフィールド数から1を引いた数になります。 これは、ターゲットフィールドの内容を後続のSEPARATEステートメントで再度分割するときなどに有用です。 次に例を示します。 `COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH ALL DELIMITERS '*'` `Content of #TARGET is: A**C*`

処理

フォーマットBの転送先フィールドは、フォーマットAの転送先フィールドと同様に処理されます。

COMPRESS操作は、全オペランドの処理が終了するか、またはターゲットフィールド（*operand2*）に空きがなくなると終了します。

ターゲットフィールドが全オペランドの合計桁数よりも大きい場合、*operand2*の残りの部分は空白となります。 ターゲットフィールドの方が小さい場合、値は切り捨てられます。

*operand2*がDYNAMIC変数である場合、COMPRESS操作は全ソースオペランドが処理されると終了します。 値の切り捨ては行われません。 COMPRESS操作後の*operand2*の長さは、ソースオペランドの合計の長さに対応します。ダイナミック変数の現在の長さは、システム変数*LENGTHを使用して確認できます。

例

このsectionでは、次のトピックについて説明します。

- 例 1 - COMPRESS
- 例 2 - COMPRESS LEAVING NO SPACE

例 1 - COMPRESS

```
** Example 'CMPEX1': COMPRESS
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 MIDDLE-I
*
1 #COMPRESSED-NAME (A20)
END-DEFINE
*
LIMIT 4
READ EMPLOY-VIEW BY NAME
  COMPRESS FIRST-NAME MIDDLE-I NAME INTO #COMPRESSED-NAME
  DISPLAY NOTITLE
          FIRST-NAME MIDDLE-I NAME 5X #COMPRESSED-NAME
END-READ
*
END
```

プログラム **CMPEX1** の出力：

```
     FIRST-NAME      MIDDLE-I         NAME               #COMPRESSED-NAME
-------------------- -------- --------------------     --------------------

KEPA                          ABELLAN                  KEPA ABELLAN
ROBERT               W        ACHIESON                 ROBERT W ACHIESON
SIMONE                        ADAM                     SIMONE ADAM
JEFF                 H        ADKINSON                 JEFF H ADKINSON
```

例 2 - COMPRESS LEAVING NO SPACE

```
** Example 'CMPEX2': COMPRESS (with LEAVING NO SPACE)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CURR-CODE (1)
  2 SALARY    (1)
*
1 #CCSALARY   (A20)
END-DEFINE
*
```

```
LIMIT 4
READ EMPL-VIEW BY NAME

  COMPRESS CURR-CODE (1) SALARY (1) INTO #CCSALARY
           LEAVING NO SPACE
  DISPLAY NOTITLE
          NAME CURR-CODE (1) SALARY (1) 5X #CCSALARY
END-READ
*
END
```

プログラム **CMPEX2** の出力：

```
        NAME          CURRENCY   ANNUAL            #CCSALARY
                        CODE     SALARY
-------------------- -------- ----------     --------------------

ABELLAN              PTA         1450000     PTA1450000
ACHIESON             UKL           11300     UKL11300
ADAM                 FRA          159980     FRA159980
ADKINSON             USD           34500     USD34500
```

例 3 - COMPRESS WITH DELIMITERS

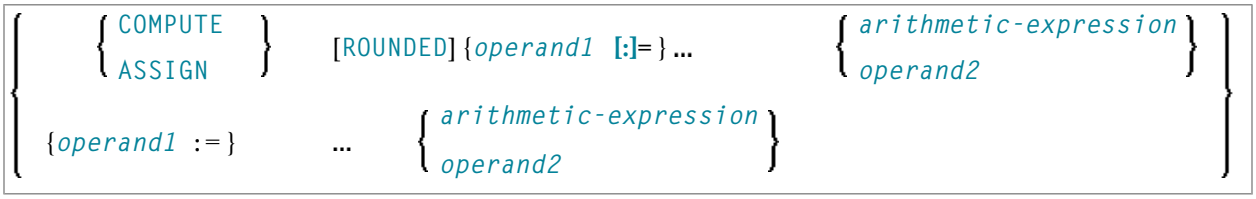
```
** Example 'CMPEX3': COMPRESS (with delimiter)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CURR-CODE (1)
  2 SALARY    (1)
*
1 #CCSALARY   (A20)
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY NAME
  COMPRESS CURR-CODE (1) SALARY (1) INTO #CCSALARY
           WITH DELIMITER '*'
  DISPLAY NOTITLE NAME CURR-CODE (1) SALARY (1) 5X #CCSALARY
END-READ
*
END
```

プログラム **CMPEX3** の出力：

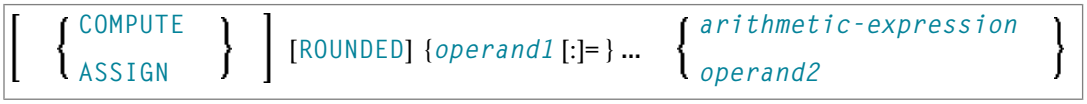
```
        NAME          CURRENCY    ANNUAL             #CCSALARY
                        CODE      SALARY
--------------------- -------- ----------     --------------------

ABELLAN               PTA         1450000     PTA*1450000
ACHIESON              UKL           11300     UKL*11300
ADAM                  FRA          159980     FRA*159980
ADKINSON              USD           34500     USD*34500
```

24 COMPUTE

ストラクチャードモード構文



レポーティングモード構文



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ADD** | **COMPRESS** | **DIVIDE** | **EXAMINE** | **MOVE** | **MOVE ALL** | **MULTIPLY** | **RESET** | **SEPARATE** | **SUBTRACT**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

COMPUTE ステートメントは、算術演算や割り当て操作で使用します。

COMPUTE ステートメントで複数のターゲットオペランド（*operand1*）を指定しても、ソースオペランド（*operand2*）が算術演算式はない場合は、個別の COMPUTE ステートメントと同じになります。

```
#TARGET1 := #TARGET2 := #SOURCE
```

これは、以下と同じです。

```
#TARGET1 := #SOURCE
#TARGET2 := #SOURCE
```

例：

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:3) INIT <3,0,9>
1 #INDEX(I4)
1 #RESULT(I4)
END-DEFINE
*
#INDEX := 1
*
#INDEX :=       /* #INDEX is 3
#RESULT :=      /* #RESULT is 9
#ARRAY(#INDEX)
*
#INDEX := 2
*
#INDEX  :=      /* #INDEX is 0
#ARRAY(3) :=    /* returns run time error NAT1316
#ARRAY(#INDEX)
END
```

ソースオペランドが算術演算式の場合は、式が評価され、その結果は一時変数に格納されます。その後、一時変数はターゲットオペランドに割り当てられます。

```
#TARGET1 := #TARGET2 := #SOURCE1 + 1
is identical to
#TEMP := #SOURCE1 + 1
#TARGET1 := #TEMP
#TARGET2 := #TEMP
```

例：

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:3) INIT <2, 0, 9>
1 #INDEX(I4)
1 #RESULT(I4)
END-DEFINE
*
#INDEX := 1
*
#INDEX :=   /* #INDEX is 3
#RESULT :=   /* #RESULT is 3
#ARRAY(#INDEX) + 1
*
#INDEX := 2
*
#INDEX  :=  /* #INDEX is 0
```

```
#ARRAY(3) :=   /* returns run time error NAT1316
#ARRAY(#INDEX)
END
```

詳細については、『*プログラミングガイド*』の「*演算割り当てのルール*」、および特に次のセクションを参照して下さい。

- *配列での算術演算*
- 「*データ転送*」（データ転送の互換性およびデータ転送のルールに関する情報）

構文説明

オペランド定義テーブル：

オペランド	構文要素						フォーマット													参照	ダイナミック定義
operand1		S	A		M		A	U	N	P	I	F	B	D	T	L	C	G	O	可	可
operand2	C	S	A		N	E	A	U	N	P	I	F	B	D	T	L	C	G	O	可	不可

構文要素の説明：

COMPUTE ASSIGN [:]=	このステートメントは、ステートメントキーワード COMPUTE（または ASSIGN）を省略して短縮形で発行することができます。 ストラクチャードモードでステートメントキーワード COMPUTE（または ASSIGN）を省略する場合には、割り当て文字（=）の前にコロン（:）を指定します。 ただし、ROUNDED オプションを使用する場合は、ステートメント名である COMPUTE または ASSIGN を指定する必要があります。
ROUNDED	キーワード ROUNDED を指定すると、値は *operand1* に割り当てられる前に切り上げられます。切り上げの詳細については、『*プログラミングガイド*』の「*演算割り当てのルール*」、および「*フィールドの切り捨てと切り上げ*」を参照してください。
operand1	結果フィールド： *operand1* には、算術／割り当て演算の結果が格納されます。 結果の精度については、『*プログラミングガイド*』の「*算術演算結果の精度*」を参照してください。 *operand1* がデータベースフィールドでも、データベースのそのフィールドは更新されません。 *operand1* がダイナミック変数の場合は、*operand2* の長さまたは算術演算の結果の長さまで値が入り、それに応じて *operand1* の長さは調整されま

	す。 ダイナミック変数の現在の長さは、システム変数 *LENGTH を使用して確認できます。 ダイナミック変数の全般的な情報については、「*ラージ変数／フィールドとダイナミック変数／フィールド*」を参照してください。
arithmetic-expression	算術演算式は、1つ以上の定数、データベースフィールド、およびユーザー定義変数で構成されます。 Natural 算術関数（『*システム関数*』の算術関数を参照）は、算術オペランドとしても使用できます。 算術演算式のオペランドは、N、P、I、F、D、または T フォーマットで定義する必要があります。 オペランドのフォーマットについては、『*プログラミングガイド*』の「*フォーマット混合式のパフォーマンスについて*」を参照してください。 次の結合演算子が使用できます。 **演算子** / **記号** カッコ / ( ) 累乗 / ** 乗算 / * 除算 / / 加算 / + 減算 / - 各演算子の前後には少なくとも1つの空白を挿入して、上記の文字を持つ変数名と混同しないようにしてください。 算術演算の処理順序は次のとおりです。 1. カッコ 2. 累乗 3. 乗算／除算（現れた順に左から右へ） 4. 加算／減算（現れた順に左から右へ）
operand2	ソースフィールド： *operand2* はソースフィールドです。 *operand1* がフォーマット C の場合は、*operand2* を属性定数として指定することもできます（『*プログラミングガイド*』の「*ユーザー定義定数*」を参照）。

除算結果の精度

COMPUTE ステートメントの除算結果の精度（桁数）は、1 番目のオペランド（被除数）または最初の結果フィールドのどちらか大きい方の精度により決まります。

ただし、整数オペランドの除算には、次のことが適用されます。2つの整定数の除算について、その結果の精度は最初の結果フィールドの精度により決まりますが、2 つの整数オペランドの 1 つでも変数であれば、結果も整数フォーマットになります。つまり、結果フィールドの精度に関わらず小数桁なしになります。

SUBSTRING オプション

オペランドが英数字、Unicode、またはバイナリフォーマットの場合は、SUBSTRING オプションを使用して *operand2* の一部分を *operand1* に割り当てることができます。記述方法は MOVE ステートメントと同じです。

例

- 例1 - ASSIGN ステートメント
- 例2 - COMPUTE ステートメント

例1 - ASSIGN ステートメント

```
** Example 'ASGEX1S': ASSIGN (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 #A (N3)
1 #B (A6)
1 #C (NO.3)
1 #D (NO.5)
1 #E (N1.3)
1 #F (N5)
1 #G (A25)
1 #H (A3/1:3)
END-DEFINE
*
ASSIGN #A = 5                          WRITE NOTITLE '=' #A
ASSIGN #B = 'ABC'                      WRITE '=' #B
ASSIGN #C = .45                        WRITE '=' #C
ASSIGN #D = #E = -0.12345              WRITE '=' #D / '=' #E
ASSIGN ROUNDED #F = 199.999            WRITE '=' #F
```

```
#G     := 'HELLO'                          WRITE '=' #G
#H (1) := 'UVW'
#H (3) := 'XYZ'                            WRITE '=' #H (1:3)
*
END
```

プログラム **ASGEX1S** の出力：

```
#A:    5
#B: ABC
#C:  .450
#D: -.12345
#E: -0.123
#F:    200
#G: HELLO
#H: UVW     XYZ
```

レポーティングモードの例については、ライブラリ SYSEXRM のプログラム **ASGEX1R** を参照してください。

例2 - COMPUTE ステートメント

```
** Example 'CPTEX1': COMPUTE
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 SALARY    (1:2)
*
1 #A          (P4)
1 #B          (N3.4)
1 #C          (N3.4)
1 #CUM-SALARY (P10)
1 #I          (P2)
END-DEFINE
*
COMPUTE #A = 3 * 2 + 4 / 2 - 1
WRITE NOTITLE 'COMPUTE #A = 3 * 2 + 4 / 2 - 1' 10X '=' #A
*
COMPUTE ROUNDED #B = 3 -4 / 2 * .89
WRITE 'COMPUTE ROUNDED #B = 3 -4 / 2 * .89' 5X '=' #B
*
COMPUTE #C = SQRT (#B)
WRITE 'COMPUTE #C = SQRT (#B)' 18X '=' #C
*
LIMIT 1
READ EMPLOY-VIEW BY PERSONNEL-ID STARTING FROM '20017000'
  WRITE / 'CURRENT SALARY: '  4X SALARY (1)
```

```
       / 'PREVIOUS SALARY:'  4X SALARY (2)
  FOR #I = 1 TO 2
    COMPUTE #CUM-SALARY = #CUM-SALARY + SALARY (#I)
  END-FOR
  WRITE 'CUMULATIVE SALARY:' #CUM-SALARY
END-READ
*
END
```

プログラム **CPTEX1** の出力：

```
COMPUTE #A = 3 * 2 + 4 / 2 - 1          #A:     7
COMPUTE ROUNDED #B = 3 -4 / 2 * .89     #B:    1.2200
COMPUTE #C = SQRT (#B)                  #C:    1.1045

CURRENT SALARY:          34000
PREVIOUS SALARY:         32300
CUMULATIVE SALARY:       66300
```

25 CREATE OBJECT

```
CREATE OBJECT operand1 OF [CLASS] operand2
  [ON [NODE] operand3]
  [GIVING operand4]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DEFINE CLASS** | **INTERFACE** | **METHOD** | **PROPERTY** | **SEND METHOD**

関連機能グループ：「*コンポーネントベースプログラミング*」

機能

`CREATE OBJECT` ステートメントは、クラスのインスタンスを作成します。

`CREATE OBJECT` ステートメントを Windows 環境で実行すると、ステートメントに指定したクラスの名前が DCOM クラスとして登録されているかどうかが Natural でチェックされます。 登録されている場合は、DCOM を使用してオブジェクトを作成します。 登録されていない場合は、現在の Natural ライブラリまたは STEPLIB 内でその名前のクラスを検索し、オブジェクトをローカルに作成します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												オペランド参照	ダイナミック定義
operand1		S															O	不可	不可
operand2	C	S				A												可	不可
operand3	C	S				A												可	不可
operand4		S			N					I								可	不可

構文要素の説明：

operand1	オブジェクトハンドル： *operand1*は、オブジェクトハンドル（HANDLE OF OBJECT）として定義する必要があります。 オブジェクトが正常に作成されると、オブジェクトハンドルには値が挿入されます。 値が正常に返されなかった場合、*operand1* の値は NULL-HANDLE になります。
OF **CLASS***operand2*	クラス名： *operand2* は、作成するオブジェクトのクラスの名前です。 DCOM クラスとして登録されていないクラスについては、DEFINE CLASS ステートメントで定義したクラス名を割り当てる必要があります。 DCOM クラスとして登録されている名前については、クラスの ProgID またはクラス GUID のどちらかを割り当てる必要があります。 DCOM クラスとして登録される Natural クラスの場合、ProgID は DEFINE CLASS ステートメントに指定されたクラス名に対応しています。 詳細については、「*Natural での登録*」を参照してください。 `CREATE OBJECT #O1 OF CLASS "Employee" or` `CREATE OBJECT #O1 OF CLASS "653BCFE0-84DA-11D0-BEB3-10005A66D231"`
ON NODE *operand3*	ノード： *operand3* には、オブジェクトが作成されるノードを指定します。 これは、クラスが DCOM クラスとして登録される場合にのみ可能です。 NODE 節を指定すると、そのノード上でオブジェクトの作成が試行されます。 NODE 節を指定しないか、または値が空白の場合、そのクラスのキー "RemoteServerName" の下にあるシステムレジストリで指定されているノード上に、オブジェクトが作成されます。 このレジストリキーが指定されていない場合、オブジェクトはローカルの Natural セッション内に作成されます。 次に例を示します。 `CREATE OBJECT #O1 OF CLASS "Employee" ON NODE "volcano.iceland.com"`
GIVING *operand4*	**GIVING** 節： GIVING 節を指定した場合、*operand4* の値は、エラー発生時は Natural メッセージ番号、成功時はゼロのいずれかになります。 GIVING 節を指定しない場合は、エラー発生時に Natural ランタイムエラー処理がトリガされます。

26 DECIDE FOR

```
DECIDE FOR      { FIRST }      CONDITION
               { EVERY }
 {WHEN logical-condition statement ...} ...
 [WHEN ANY statement ...]
 [WHEN ALL statement ...]
 WHEN NONE statement ...
END-DECIDE
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DECIDE ON** | **IF** | **IF SELECTION** | **ON ERROR**

関連機能グループ：「*論理条件の処理*」

機能

DECIDE FOR ステートメントは、複数の条件（ケース）に応じて 1 つ以上のアクションを実行するかどうかを判定します。

Note: ある条件のもとでは*何も実行しない*場合は、DECIDE FOR ステートメントの対応する節にステートメント IGNORE を指定する必要があります。

構文説明

FIRST CONDITION	最初に真になった条件のみが処理されます。「*例 1*」も参照してください。
EVERY CONDITION	真になった条件がすべて処理されます。「*例 2*」も参照してください。
WHEN *logical-condition* *statement*	この節では、処理される論理条件（複数可）を指定します。『*プログラミングガイド*』の「*論理条件基準*」を参照してください。
WHEN ANY *statement*	WHEN ANY では、任意の論理条件が真になったときに実行されるステートメント（複数可）を指定できます。
WHEN ALL *statement*	WHEN ALL では、すべての論理条件が真になったときに実行されるステートメント（複数可）を指定できます。 この節は、EVERY が指定された場合にのみ適用されます。
WHEN NONE *statement*	WHEN NONE では、真になっている論理条件が 1 つもないときに実行されるステートメント（複数可）を指定できます。

END-DECIDE	DECIDE FOR ステートメントを終了するには、Natural 予約語 END-DECIDE を使用する必要があります。

例

例 1 - FIRST オプションを指定した DECIDE FOR

```
** Example 'DECEX1': DECIDE FOR (with FIRST option)
************************************************************************
DEFINE DATA LOCAL
1 #FUNCTION (A1)
1 #PARM     (A1)
END-DEFINE
*
INPUT #FUNCTION #PARM
*
DECIDE FOR FIRST CONDITION
  WHEN #FUNCTION = 'A' AND #PARM = 'X'
    WRITE 'Function A with parameter X selected.'
  WHEN #FUNCTION = 'B' AND #PARM = 'X'
    WRITE 'Function B with parameter X selected.'
  WHEN #FUNCTION = 'C' THRU 'D'
    WRITE 'Function C or D selected.'
  WHEN NONE
    REINPUT 'Please enter a valid function.'
            MARK *#FUNCTION
END-DECIDE
*
END
```

プログラム **DECEX1** の出力：

```
#FUNCTION A #PARM Y
```

Enter キーを押した後：

```
PLEASE ENTER A VALID FUNCTION
#FUNCTION A #PARM Y
```

例 2 - EVERY オプションを指定した DECIDE FOR

```
** Example 'DECEX2': DECIDE FOR (with EVERY option)
************************************************************************
DEFINE DATA LOCAL
1 #FIELD1 (N5.4)
END-DEFINE
*
INPUT #FIELD1
*
DECIDE FOR EVERY CONDITION
  WHEN #FIELD1 >= 0
    WRITE '#FIELD1 is positive or zero.'
  WHEN #FIELD1 <= 0
    WRITE '#FIELD1 is negative or zero.'
  WHEN FRAC(#FIELD1) = 0
    WRITE '#FIELD1 has no decimal digits.'
  WHEN ANY
    WRITE 'Any of the above conditions is true.'
  WHEN ALL
    WRITE '#FIELD1 is zero.'
  WHEN NONE
    IGNORE
END-DECIDE
*
END
```

プログラム **DECEX2** の出力：

```
#FIELD1 42
```

Enter キーを押した後：

```
Page      1                                              05-01-11  14:56:26

#FIELD1 is positive or zero.
#FIELD1 has no decimal digits.
Any of the above conditions is true.
```

27 DECIDE ON

```
DECIDE ON
 { FIRST } [VALUES] [OF]
 { EVERY }
                 { operand1                                    }
                 { SUBSTRING (operand3,operand5,operand6)      }
 { VALUES        { operand2                                    }
                 { SUBSTRING (operand4,operand7,operand8)      }
               [[,{ operand2                                   }] ...
                 { SUBSTRING (operand4,operand7,operand8)      }
               [: { operand2                                   } ]]   statement ...}
                 { SUBSTRING (operand4,operand7,operand8)      }
 [ANY [VALUES] statement ...]
 [ALL [VALUES] statement ...]
 NONE [VALUES] statement ...
END-DECIDE
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DECIDE FOR** | **IF** | **IF SELECTION** | **ON ERROR**

関連機能グループ：「*論理条件の処理*」

機能

DECIDE ON ステートメントでは、変数の値（1 つまたは複数の値）に応じて行う複数の処理を指定します。

Note: ある条件のもとでは*何も実行しない*場合は、DECIDE ON ステートメントの対応する節にステートメント IGNORE を指定する必要があります。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												オペランド参照	ダイナミック定義
operand1		S	A		N	A	U	N	P	I	F	B	D	T	L		G	O	可	不可
operand2	C	S	A			A	U	N	P	I	F	B	D	T	L		G	O	可	不可
operand3		S	A		N	A	U					B							可	不可
operand4	C	S	A			A	U					B							可	不可
operand5	C	S						N	P	I		B *							可	不可
operand6	C	S						N	P	I		B *							可	不可
operand7	C	S						N	P	I		B *							可	不可
operand8	C	S						N	P	I		B *							可	不可

* *operand5* のフォーマット B、*operand6*、*operand7*、および *operand8* は、4 以下の長さでのみ使用できます。

構文要素の説明：

FIRST/EVERY	これらのキーワードの 1 つで、見つかった最初の値に対してだけ処理を行うか、または見つかったすべての値に対して処理を行うかを指定します。
operand1	選択フィールド： *operand1* または *operand2* では、内容がチェックされるフィールドの名前を指定します。
VALUES *operand2* **[[**,*operand2*] ... **[:***operand2***]***statement* ...	この節では、選択フィールドの値（*operand2*）、およびフィールドがその値になった場合に実行される *statement(s)* を指定します。 選択フィールドには、1つの値、複数の値、またはオプションで1つ以上の値の範囲を指定できます。 複数の値を指定する場合は、INPUT 区切り文字（ID セッションパラメータで指定した文字）またはコンマ（,）で区切る必要があります。セッションパラメータ DC でコンマを小数点文字として定義している場合は、この目的でコンマを使用することはできません。 値の範囲を指定する場合、範囲の開始値と終了値をコロン（:）で区切って指定します。
SUBSTRING **(***operand3*,*operand5*,*operand6*)	SUBSTRING オプションの指定がないと、フィールドの内容全体がチェックされます。 SUBSTRING オプションを指定すると、英数字

	フィールド、Unicode フィールド、またはバイナリフィールドの特定の一部分だけをチェックできます。 フィールド名（*operand3*）の後に、まず開始位置（*operand5*）を指定してから、チェック対象のフィールドの部分の長さ（*operand6*）を指定してください。
SUBSTRING (*operand4*,*operand7*,*operand8*)	フィールド名（*operand4*）の後に、まず開始位置（*operand7*）を指定してから、チェック対象のフィールドの部分の長さ（*operand8*）を指定してください。
ANY *statement*	ANY では、VALUES 節にある値のいずれかが見つかった場合に実行される *statement(s)* を指定します。 VALUES 節で指定されたステートメントに加えて、これらのステートメントが実行されます。
ALL *statement*	ALL では、VALUES 節にある値がすべて見つかった場合に実行される *statement(s)* を指定します。 VALUES 節で指定されたステートメントに加えて、これらのステートメントが実行されます。 ALL 節は、キーワード EVERY を指定した場合にのみ適用されます。
NONE *statement*	NONE では、指定した値が 1 つも見つからない場合に実行される *statement(s)* を指定します。
END-DECIDE	DECIDE ON ステートメントを終了するには、Natural 予約語 END-DECIDE を使用する必要があります。

例

例 1 - FIRST オプションを指定した DECIDE ON

```
** Example 'DECEX3': DECIDE ON (with FIRST option)
************************************************************************
*
SET KEY ALL
INPUT 'Enter any PF key' /
      'and check result' /
*
DECIDE ON FIRST VALUE OF *PF-KEY
  VALUE 'PF1'
    WRITE 'PF1 key entered.'
  VALUE 'PF2'
    WRITE 'PF2 key entered.'
  ANY VALUE
    WRITE 'PF1 or PF2 key entered.'
  NONE VALUE
```

```
    WRITE 'Neither PF1 nor PF2 key entered.'
END-DECIDE
*
END
```

プログラム **DECEX3** の出力：

```
 Enter any PF key
 and check result
```

PF1 を押した後の出力：

```
Page      1                                                  05-01-11  15:08:50

PF1 key entered.
PF1 or PF2 key entered.
```

例 2 - EVERY オプションを指定した DECIDE ON

```
** Example 'DECEX4': DECIDE ON (with EVERY option)
************************************************************************
DEFINE DATA LOCAL
1 #FIELD (N1)
END-DEFINE
*
INPUT 'Enter any value between 1 and 9:' #FIELD (SG=OFF)
*
DECIDE ON EVERY VALUE OF #FIELD
  VALUE 1 : 4
    WRITE 'Content of #FIELD is 1-4'
  VALUE 2 : 5
    WRITE 'Content of #FIELD is 2-5'
  ANY VALUE
    WRITE 'Content of #FIELD is 1-5'
  ALL VALUE
    WRITE 'Content of #FIELD is 2-4'
  NONE VALUE
    WRITE 'Content of #FIELD is not 1-5'
    END-DECIDE
*
END
```

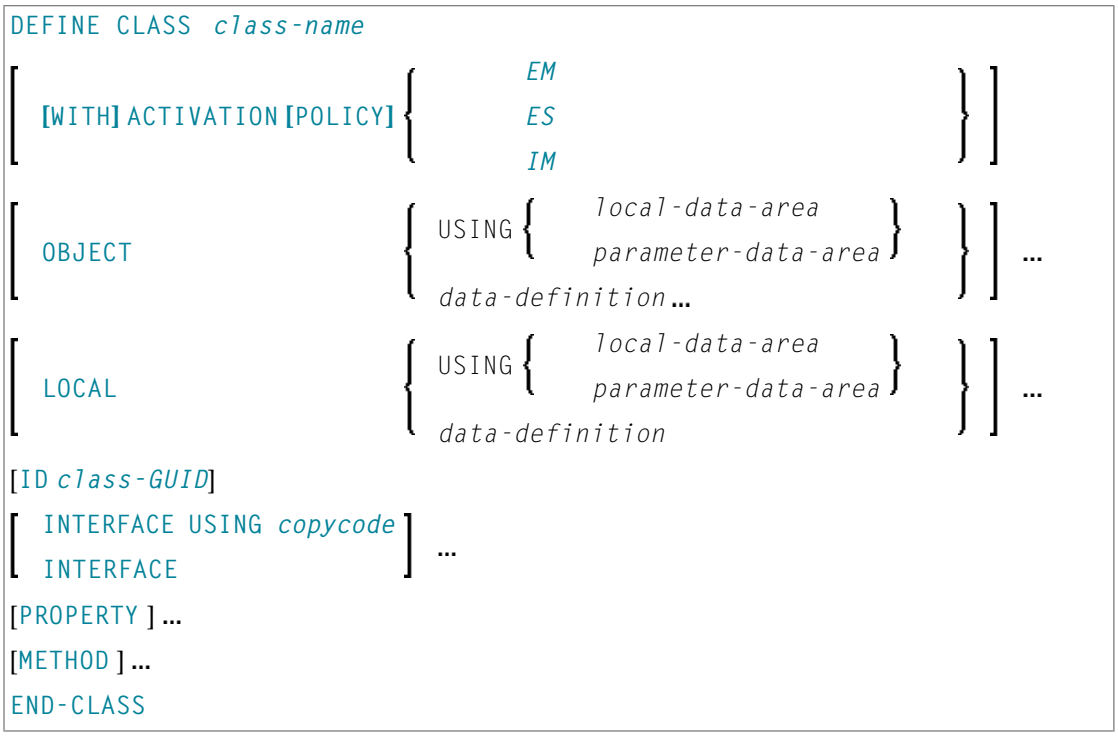
プログラム **DECEX4** の出力：

```
ENTER ANY VALUE BETWEEN 1 AND 9: 4
```

"**4**" を入力して確認した後：

```
Page      1                                                  05-01-11  15:11:45

Content of #FIELD is 1-4
Content of #FIELD is 2-5
Content of #FIELD is 1-5
Content of #FIELD is 2-4
```

28 DEFINE CLASS



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CREATE OBJECT** | **INTERFACE** | **METHOD** | **PROPERTY** | **SEND METHOD**

関連機能グループ：「*コンポーネントベースプログラミング*」

機能

DEFINE CLASS ステートメントは、Natural クラスモジュール内からクラスを指定するために使用します。 Natural クラスモジュールは、DEFINE CLASS ステートメントとそれに続く END ステートメントで構成されています。

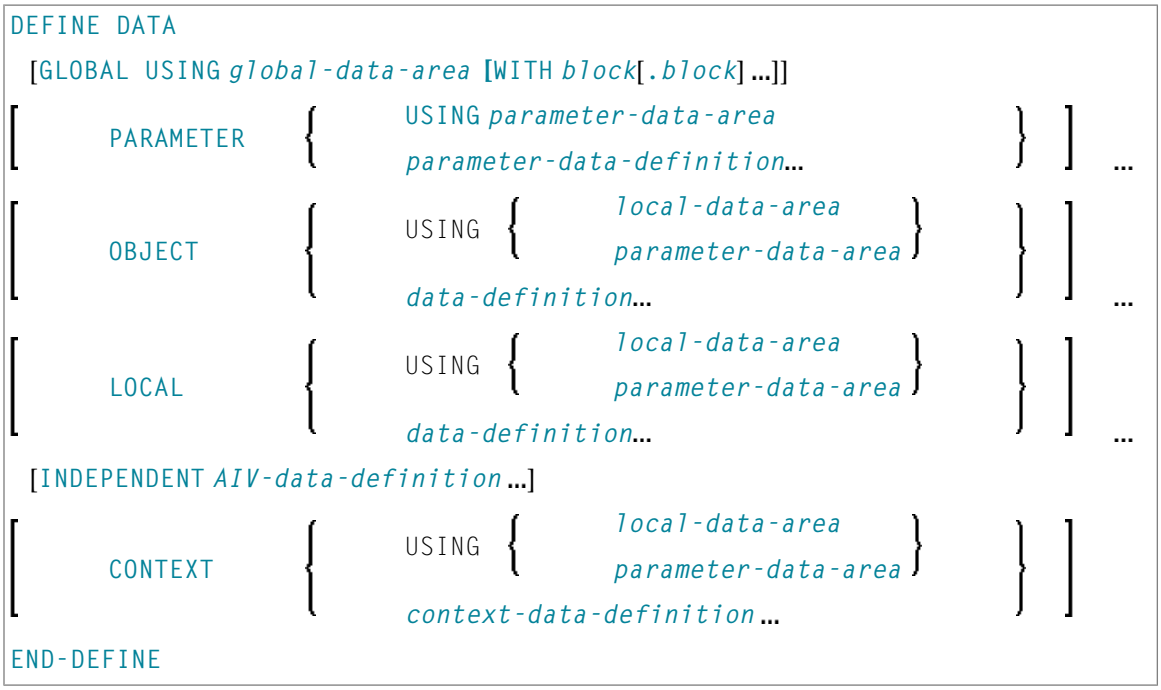
構文説明

class-name	これは、このクラスのオブジェクトを作成するためにクライアントが使用する名前です。名前の長さは、最大32文字までです。名前にはピリオドが含まれていてもかまいません。ピリオドを使用して、クラス名を次のように構成できます。 *company-name*. *application-name*. *class-name* ピリオド（...）間にある各部分は、それぞれがユーザー変数の命名規則に従っている必要があります（「ユーザー定義変数の命名規則」を参照）。 異なるプログラミング言語で書かれたクライアントによってクラスを使用する予定であれば、クラス名はこれらの言語に適用する命名規則と矛盾しないように選択する必要があります。	
WITH ACTIVATION POLICY	WITH ACTIVATION POLICY 節は、現在のクラスに登録されるアクティベーションポリシーを明示的に定義するために使用します。 以下のパラメータを設定できます。	
	EM	アクティベーションポリシー「ExternalMultiple」 を設定します。
	ES	アクティベーションポリシー「ExternalSingle」 を設定します。
	IM	アクティベーションポリシー「InternalMultiple」 を設定します。
	クラスが保存（stow）および登録されるときに、WITH ACTIVATION POLICY 節の設定によってACTPOLICY（アクティベーションポリシー）プロファイルパラメータが上書きされますが、アクティベーションポリシーを明示的に定義する REGISTER コマンドを使用して手動で登録することでも上書きされます。 詳細については、『*オペレーション*』ドキュメントの「*アクティベーションポリシー*」を参照してください。	
OBJECT	OBJECT 節は、オブジェクトデータを定義するために使用します。 OBJECT 節の構文は、DEFINE DATA ステートメントの LOCAL 節と同じです。 詳細については、DEFINE DATA ステートメントの LOCAL 節に関する説明を参照してください。	
LOCAL	LOCAL 節は、グローバルユニーク ID（GUID）をクラス定義に含めるためにのみ使用します。 GUID は、クラスが DCOM で登録される場合にのみ定義する必要があります。 GUID は、主にローカルデータエリア内で定義されます。 詳細については、『*プログラミングガイド*』の「*グローバルユニーク ID - GUID*」を参照してください。 LOCAL 節の構文は、DEFINE DATA ステートメントの LOCAL 節と同じです。 詳細については、DEFINE DATA ステートメントの LOCAL 節に関する説明を参照してください。	
ID	ID 節は、グローバルユニーク ID をクラスに割り当てるために使用します。 クラス GUID は、LOCAL 節に含まれるデータエリア内に定義される GUID の名前です。 クラス GUID は、（命名された）英数字の定数です。 クラスが DCOM で登録される場合は、GUID をクラスに割り当てる必要があります。	
INTERFACE USING	INTERFACE USING 節は、INTERFACE ステートメントを含むコピーコードを組み込むために使用します。	

copycode	INTERFACE USING 節で使用するコピーコードには、1 つ以上の INTERFACE ステートメントを含めることができます。
PROPERTY	PROPERTY ステートメントは、インターフェイス定義外で、実装としてオブジェクトデータ変数オペランドをプロパティに割り当てます。
METHOD	METHOD ステートメントは、インターフェイス定義外で、サブプログラムをメソッドに割り当てます。
END-CLASS	DEFINE CLASS ステートメントを終了するには、Natural 予約語 END-CLASS を使用する必要があります。

29 DEFINE DATA

全般的な構文



DEFINE DATA ステートメントは、ローカルデータエリア（LDA）、グローバルデータエリア（GDA）またはパラメータデータエリア（PDA）に含まれる定義済みのデータ定義を参照するか、インライン定義を書き込むことにより、Naturalプログラムで使用するデータ定義を宣言する多数の節を提供します。

DEFINE DATA ステートメントに関するドキュメントは、次のセクションに分かれています。

- 構文の概要
- **DEFINE DATA -** 全般

特定のデータ定義：

- ローカルデータの定義
- グローバルデータの定義
- パラメータデータの定義
- アプリケーションに依存しない変数の定義
- **Natural RPC** 用のコンテキスト変数の定義
- **NaturalX** オブジェクトの定義

節とオプション：

- 変数定義
- ビューの定義
- 再定義
- ハンドルの定義
- 配列の次元の定義
- 初期値の定義
- 配列用の初期値／定数値
- フィールド／変数の **EM**、**HD**、**PD** パラメータ

例：

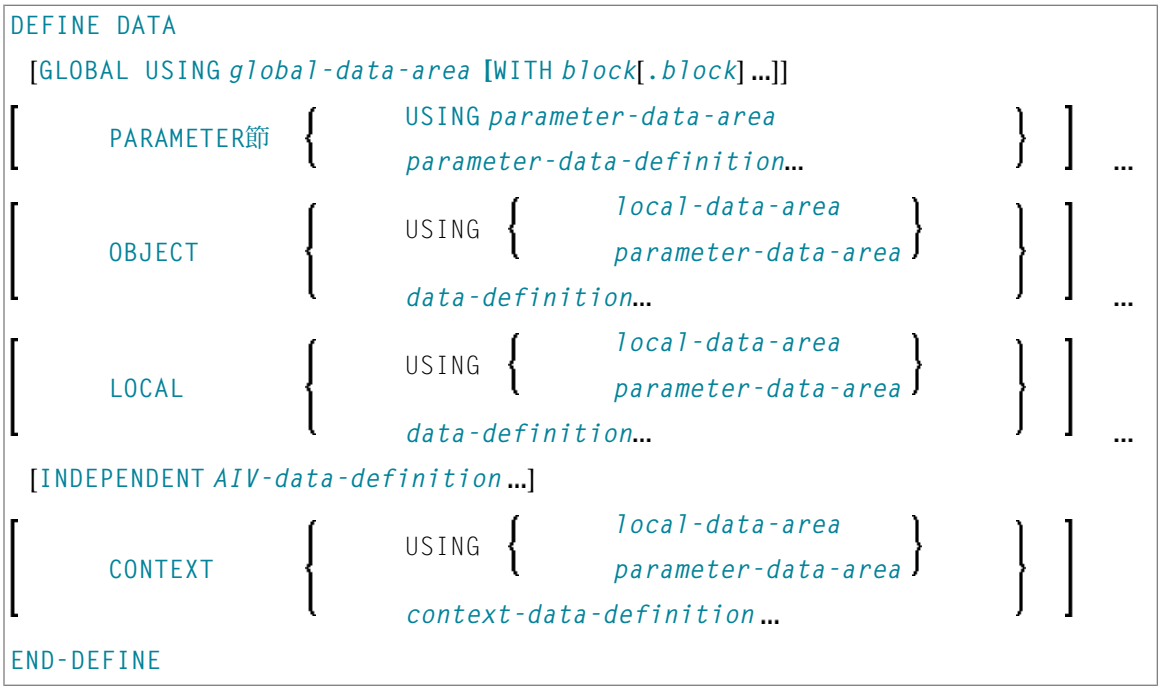
- **DEFINE DATA** ステートメントの使用例

30 構文の概要

このchapterには、DEFINE DATA ステートメントの説明で使用される構文ボックスの完全な概要が記載されています。

プログラムのステートメント行の中でキーワード、節、パラメータ、オプション、および他の構文要素を配列および組み合わせる方法について説明します。

全般的な構文



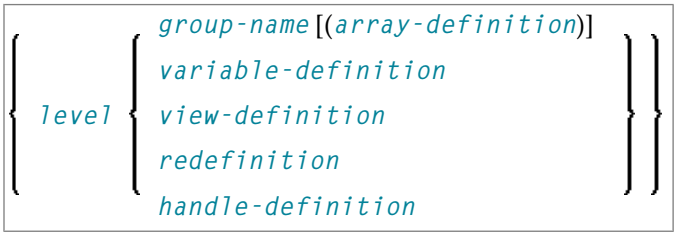
基本的な構文要素

以下では次のトピックについて説明します。

- *data-definition*
- *parameter-data-definition*
- *parameter-handle-definition*
- *variable-definition*
- *view-definition*
- *redefinition*
- *init-definition*
- *array-definition*
- *array-init-definition*
- *emhdpm*

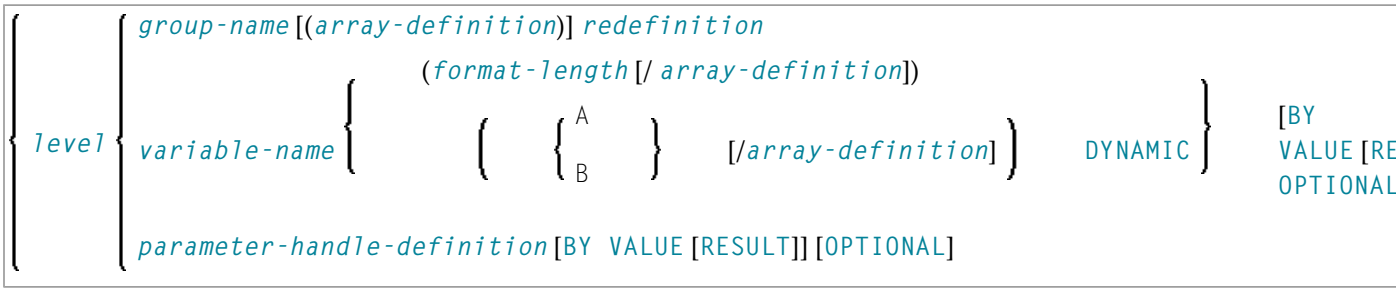
- *AIV-data-definition*
- *context-data-definition*

data-definition



詳細については、「*ローカルデータの定義*」または「***NaturalX オブジェクトの定義***」を参照してください。

parameter-data-definition

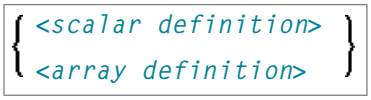


詳細については、「*パラメータデータの定義*」を参照してください。

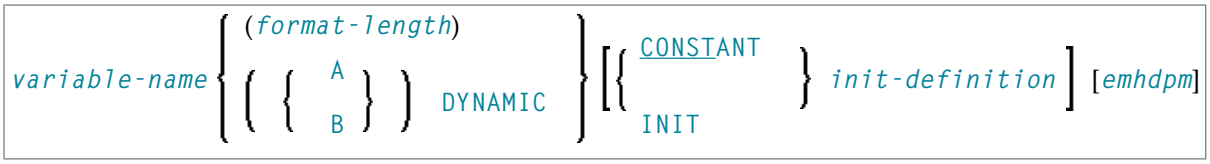
parameter-handle-definition

詳細については、*パラメータハンドルのデータ定義*に関する説明を参照してください。

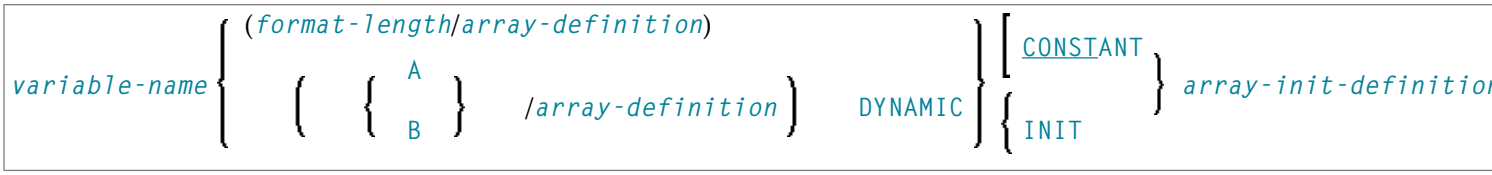
variable-definition



<scalar definition>

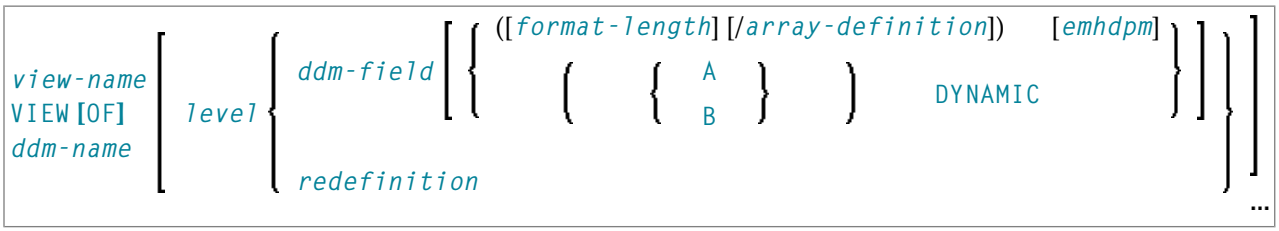


<array definition>



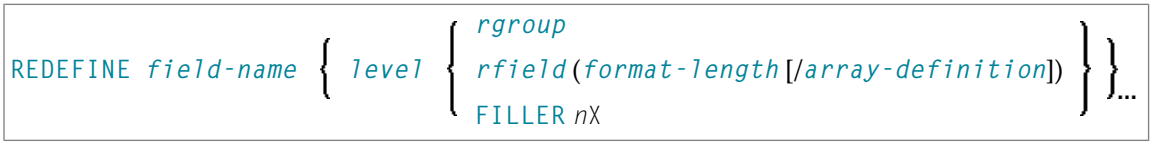
詳細については、「*変数定義*」を参照してください。

view-definition



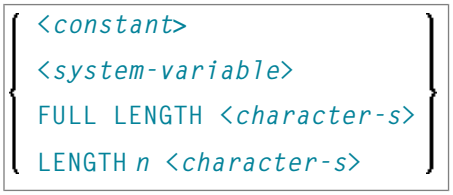
詳細については、「*ビューの定義*」を参照してください。

redefinition



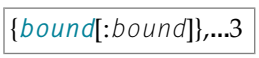
詳細については、「*再定義*」を参照してください。

init-definition



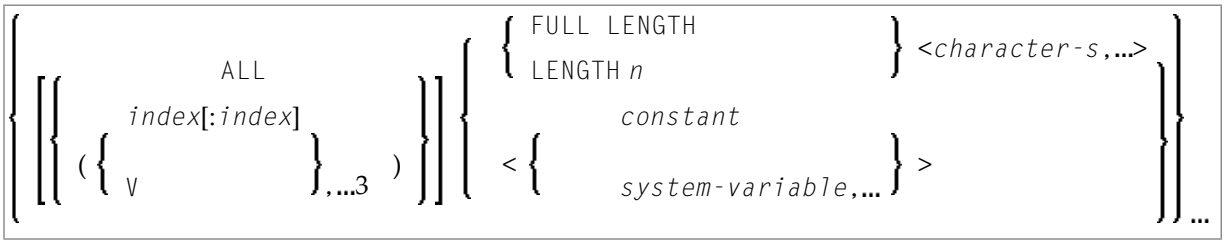
詳細については、「*配列用の初期値／定数値*」を参照してください。

array-definition



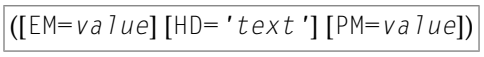
詳細については、「*配列の次元の定義*」を参照してください。

array-init-definition



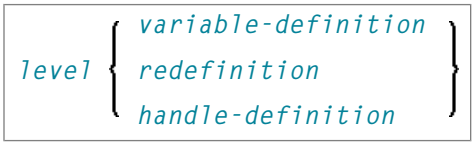
詳細については、「*配列用の初期値／定数値*」を参照してください。

emhdpm



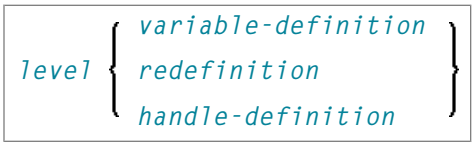
詳細については、「*フィールド／変数の* ***EM*** *、* ***HD*** *、* ***PM*** *パラメータ*」を参照してください。

AIV-data-definition



詳細については、「*アプリケーションに依存しない変数の定義*」を参照してください。

context-data-definition



詳細については、「***Natural RPC** 用のコンテキスト変数の定義*」を参照してください。

31 DEFINE DATA - 全般

このchapterでは、次のトピックについて説明します。

機能

DEFINE DATA ステートメントは、ローカルデータエリア（LDA）、グローバルデータエリア（GDA）またはパラメータデータエリア（PDA）に含まれる定義済みのデータ定義を参照するか、インライン定義を書き込むことにより、Naturalプログラムで使用するデータ定義を宣言する多数の節を提供します。

規則

- DEFINE DATA ステートメントを使用する場合は、プログラムまたはルーチンの最初のステートメントとして指定する必要があります。
- 「空の」 DEFINE DATA ステートメントを使用することはできません。つまり、少なくとも 1 つの節（LOCAL、GLOBAL、PARAMETER、INDEPENDENT、CONTEXT、または OBJECT）を指定する必要があり、また少なくとも 1 つのフィールドを定義する必要があります。
- 複数の節を指定できますが、その場合は、構文図に示されている順序で節を指定する必要があります。
- Natural 予約語 END-DEFINE を使用して、DEFINE DATA ステートメントを終了させる必要があります。

プログラミングモード

DEFINE DATA ステートメントは、ストラクチャードモードおよびレポーティングモードでで使用できます。 これらのモードでの違いについては、DEFINE DATA ステートメントの説明を参照してください。

通常、次のことが適用されます。

- ストラクチャードモード

- レポーティングモード

ストラクチャードモード

使用するすべての変数は、アプリケーションに依存しない変数（AIV）を除き、DEFINE DATA ステートメントで定義する必要があり、プログラム内の他の場所で定義することはできません。DEFINE DATA INDEPENDENT ステートメントを使用する場合は、プログラム内の他の場所で AIV を定義することはできません。

レポーティングモード

変数はプログラムの本体で定義できるため、DEFINE DATA ステートメントは必須ではありません。 ただし、DEFINE DATA LOCAL ステートメントをレポーティングモードで使用する場合は、アプリケーションに依存しない変数（AVI）を除く変数を、プログラム内の他の場所で定義することはできません。また、DEFINE DATA INDEPENDENT ステートメントを使用する場合は、アプリケーションに依存しない変数（AIV）をプログラム内の他の場所で定義することはできません。

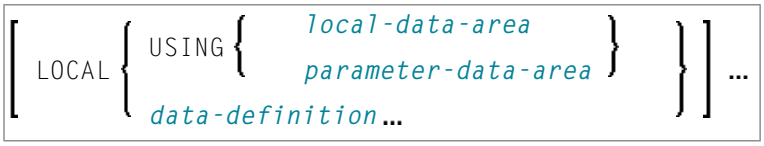
詳細な情報

DEFINE DATA ステートメントの詳細については、『*プログラミングガイド*』の次のセクションを参照してください。

- *DEFINE DATA ステートメントの使用と構造*
- *データエリアの使用*

32 ローカルデータの定義

DEFINE DATA LOCAL の一般的な構文：



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

DEFINE DATA LOCAL ステートメントでは、アプリケーション内にある単一の Natural モジュールによって排他的に使用されるデータ要素を定義します。 これらのデータ要素またはフィールドは、ステートメント自体の中で定義できます（*data-definition* を参照）。または、データエリアを参照するステートメントを使用して、プログラムの外にある別のローカルデータエリア（LDA）やパラメータデータエリア（PDA）でデータ要素またはフィールドを定義することもできます。

制限事項

LDA とそれを参照するオブジェクトは、同じライブラリ（または STEPLIB）に含まれている必要があります。

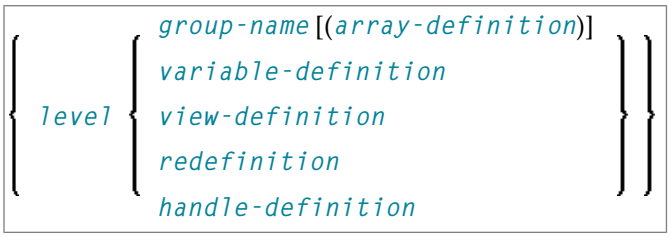
構文説明

local-data-area	ローカルデータエリアには、DEFINE DATA LOCAL ステートメントに含めることができる定義済みのデータ要素が格納されます。 複数のデータエリアを参照できますが、その場合は次の例のように、予約語 LOCAL および USING を繰り返す必要があります。 `DEFINE DATA LOCAL` `  LOCAL USING DATX_L` `  LOCAL USING DATX_P`

	... END-DEFINE ; 詳細については、『*プログラミングガイド*』の「*別のデータエリアでのフィールドの定義*」、「*ローカルデータエリア*」、および「*例 2*」を参照してください。
parameter-data-area	また、DEFINE DATA LOCAL ステートメントで参照するデータエリアは、パラメータデータエリア（PDA）でもかまいません。PDA を LDA として使用することにより、PDA と同じ構造を持つ LDA を作成する手間を省くことができます。
data-definition	下記の「*ダイレクトデータ定義*」を参照してください。
END-DEFINE	Natural 予約語 END-DEFINE を使用して、DEFINE DATA ステートメントを終了させる必要があります。

ダイレクトデータ定義

ローカルデータは、プログラムまたはルーチン内で直接定義できます。 ダイレクトデータ定義には、次の構文が適用されます。



詳細については、次を参照してください。

- *例 1 - DEFINE DATA LOCAL*（ダイレクトデータ定義）
- 『*プログラミングガイド*』の「*DEFINE DATA ステートメント内のフィールドの定義*」
- 『*プログラミングガイド*』の「*ローカルデータエリア*」と「*例 1*」

ダイレクトデータ定義の構文要素の説明

level	レベル番号は 01～99 の範囲内の 1 桁または 2 桁の数字（先頭の "0" は任意）を使用し、フィールドグルーピングと合わせて使用します。02 またはそれ以上のレベル番号が割り当てられたフィールドは、それより小さいレベル番号が割り当てられた直前のグループの一部であるとみなされます。 グループを定義すると、グループ名を使用して一連のフィールド（または 1 つのフィールドのみ）を参照できます。特定のステートメント（CALL、CALLNAT、RESET、WRITE など）の場合、グループ名をショートカットとして指定し、グループに含まれるフィールドを参照することもできます。

	グループには、他のグループを含めることができます。グループにレベル番号をつけるときは、レベル番号を飛ばしてはなりません。 ビューの定義は常にレベル 1 で定義する必要があります。
group-name	グループの名前。名前は、Natural 変数名の定義ルールに準拠していなければなりません。次のセクションも参照してください。 ■ 『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」 ■ 『*プログラミングガイド*』の「*データ構造の条件指定*」
array-definition	*array-definition* の場合、配列定義の次元の下限と上限を定義します。「*配列の次元の定義*」を参照してください。
variable-definition	*variable-definition* を使用して、単一の値（スカラ）または複数の値（配列）を持つ単一のフィールド／変数を定義します。「*変数定義*」を参照してください。
view-definition	*view-definition* を使用して、データ定義モジュール（DDM）から派生したビューを定義します。「*ビューの定義*」を参照してください。
redefinition	*redefinition* を使用して、グループ、ビュー、DDM フィールド、単一のフィールド／変数（スカラまたは配列）を再定義できます。「*再定義*」を参照してください。
handle-definition	ハンドルは、コードのダイアログエレメントを特定し、ハンドル変数に格納されます。「*ハンドルの定義*」を参照してください。

33 グローバルデータの定義

DEFINE DATA GLOBAL の一般的な構文：

```
DEFINE DATA
GLOBAL USING global-data-area [WITH block[.block...]]
END-DEFINE
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

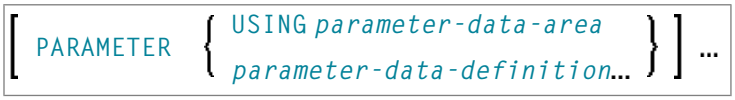
DEFINE DATA GLOBAL ステートメントでは、グローバルデータエリア（GDA）を使用してデータ要素を定義します。

構文説明

USING *global-data-area*	グローバルデータエリア（GDA）には、複数のプログラミングオブジェクトで参照できるデータ要素が含まれています。
WITH *block*	データストレージスペースを保存するために、データブロックでグローバルデータエリアを作成できます。プログラムの実行中にデータブロックが互いに重なり合うことも可能であるため、ストレージスペースの節約になります。 ブロックレベルの最大値は 8（マスタブロックを含む）です。 詳細については、『*プログラミングガイド*』の「*データブロック*」を参照してください。
.block	*.block* 表記（複数可）では、プログラム内で使用されるブロックを指定できます。
END-DEFINE	Natural 予約語 END-DEFINE を使用して、DEFINE DATA ステートメントを終了させる必要があります。

34 パラメータデータの定義

DEFINE DATA PARAMETER の一般的な構文：



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

DEFINE DATA PARAMETER ステートメントでは、Natural サブプログラム、外部サブルーチン、ヘルプルーチン、関数、またはダイアログで、受信パラメータとして使用できるデータ要素を定義します。 これらのパラメータは、ステートメント自体の中で定義できます（下記の「*パラメータデータ定義*」を参照）。または、データエリアを参照するステートメントを使用して、プログラムの外にあるパラメータデータエリア（PDA）でパラメータを定義することもできます。

制限事項

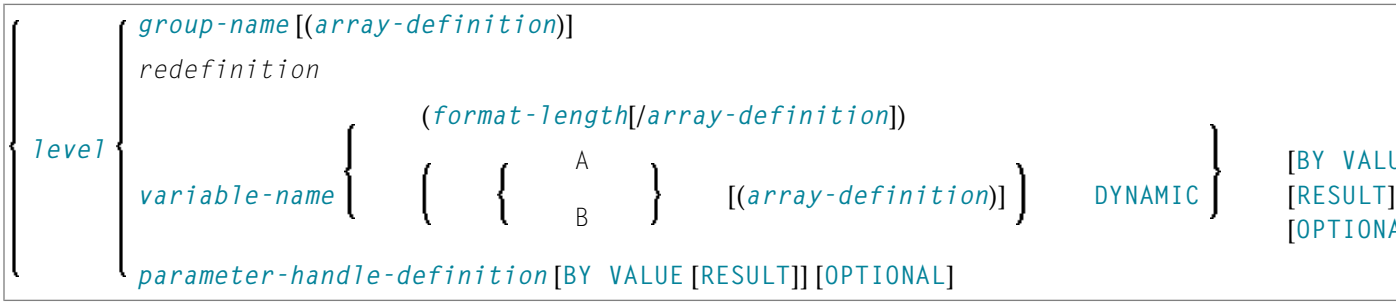
- パラメータのデータ要素には、初期値または定数値を割り当てることはできず、またこれらのデータ要素では、編集マスク（EM）、ヘッダー（HD）、または出力モード（PM）を保持することはできません（「*フィールド／変数の EM、HD、PM パラメータ*」も参照）。
- パラメータデータエリアとそれを参照するオブジェクトは、同じライブラリ（またはSTEPLIB）に含まれている必要があります。

構文説明

USING *parameter-data-area*	サブプログラム、外部サブルーチン、またはダイアログでパラメータとして使用されるデータ要素を含む *parameter-data-area* の名前。
parameter-data-definition	パラメータデータエリアを定義する代わりに、プログラムまたはサブルーチン内でパラメータデータを直接定義することもできます。下記の「*パラメータデータ定義*」を参照してください。
END-DEFINE	Natural 予約語 END-DEFINE を使用して、DEFINE DATA ステートメントを終了させる必要があります。

パラメータデータ定義

ダイレクトパラメータデータ定義には、次の構文が適用されます。



構文要素の説明：

level	レベル番号は 01～99 の範囲内の 1 桁または 2 桁の数字（先頭の "0" は任意）を使用し、フィールドグルーピングと合わせて使用します。02 またはそれ以上のレベル番号が割り当てられたフィールドは、それより小さいレベル番号が割り当てられた直前のグループの一部であるとみなされます。 グループを定義すると、グループ名を使用して一連のフィールド（または 1 つのフィールドのみ）を参照できます。 特定のステートメント（CALL、CALLNAT、RESET、WRITE など）の場合、グループ名をショートカットとして指定し、グループに含まれるフィールドを参照することもできます。 グループには、他のグループを含めることができます。 グループにレベル番号をつけるときは、レベル番号を飛ばしてはなりません。
group-name	グループの名前。名前は、Natural 変数名の定義ルールに準拠していなければなりません。 次のセクションも参照してください。 ■ 『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」 ■ 『*プログラミングガイド*』の「*データ構造の条件指定*」
array-definition	*array-definition* の場合、配列定義の次元の下限と上限を定義します。 「*配列の次元の定義*」および「*パラメータデータエリア内の可変配列*」を参照してください。
redefinition	*redefinition* を使用して、グループまたは単一のフィールド／変数（スカラまたは配列）を再定義できます。 「*再定義*」を参照してください。 **注意:** *parameter-data-definition* では、グループの「再定義」は REDEFINE ブロック内のみで行うことができます。

variable-name	変数に割り当てられる名前。 Natural 変数名のルールが適用されます。 ユーザー定義変数の命名規則については、『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。
format-length	フィールドのフォーマットおよび長さ。 ユーザー定義変数のフォーマットおよび長さの定義については、『*プログラミングガイド*』の「*ユーザー定義変数のフォーマットおよび長さ*」を参照してください。
A、U、または B	データタイプ：ダイナミック変数用の英数字（A）、Unicode（U）、またはバイナリ（B）。
DYNAMIC	パラメータは DYNAMIC として定義できます。 ダイナミック変数の処理については、「*ラージ変数／フィールドとダイナミック変数／フィールド*」を参照してください。
	コールモード： 参照による呼び出し、値による呼び出し、値による呼び出しと結果のいずれを使用するかに応じて、適切な転送メカニズムが適用されます。 詳細については、CALLNAT ステートメントを参照して下さい。
（BY VALUE を省略）	参照による呼び出し： 参照による呼び出しは、BY VALUE キーワードを省略した場合にデフォルトで有効になります。 この場合、パラメータは参照によって（つまりアドレスを介して）サブプログラムまたはサブルーチンに渡されるため、CALLNAT または PERFORM ステートメント内でパラメータとして指定するフィールドのフォーマット／長さは、サブプログラムまたはサブルーチンで呼び出される対応するフィールドと同じである必要があります。
BY VALUE	値による呼び出し： BY VALUE を指定すると、パラメータは値によってサブプログラムまたはサブルーチンに渡されます。つまり、（アドレスの代わりに）実際のパラメータ値が渡されます。 さらに、サブプログラムまたはサブルーチン内にあるフィールドのフォーマット／長さは、CALLNAT または PERFORM パラメータと同じである必要はありません。 フォーマット／長さでは、データ転送の互換性のみが必要です。 データ転送の互換性に関しては、『*プログラミングガイド*』の「*演算割り当てのルール*」セクションにある「*データ転送*」の*対応表*が適用されます。 BY VALUE を使用すると、例えば、サブプログラムまたはサブルーチンを呼び出すオブジェクトを調整しなくても、サブプログラムまたはサブルーチン内でフィールドの長さを増加させることができます（サブプログラムまたはサブルーチンの拡張によってそれが必要になった場合）。 ダイアログ用のパラメータ定義に関しては、次のことが適用されます。

	■ BY VALUE を省略した場合、ダイアログのパラメータデータエリアのインライン定義で指定されているように、パラメータはそのアドレスを介して（参照によって）転送されます。例えば、OPEN DIALOG または SEND EVENT ステートメントにあるパラメータのフォーマットおよび長さは、ダイアログのインラインパラメータデータ定義にあるパラメータのフォーマットおよび長さと一致している必要があります。 使用されるパラメータを、このイベントをトリガする SEND EVENT ステートメントで転送する場合は、イベントハンドラを開く前後および他のすべてのイベントで、参照によってパラメータを使用することができます。 ■ BY VALUE を指定する場合、パラメータは値を介して転送されるため、フォーマットと長さが一致している必要はありません。ただし、OPEN DIALOG または SEND EVENT ステートメントのパラメータでは、ダイアログのパラメータとの間でデータ転送の互換性が確保されている必要があります。
	BY VALUE の例：
	`* Program` `DEFINE DATA LOCAL` `1 #FIELDA (P5)` `...` `END-DEFINE` `...` `CALLNAT 'SUBR01' #FIELDA` `...` `* Subroutine SUBR01` `DEFINE DATA PARAMETER` `1 #FIELDB (P9) BY VALUE` `END-DEFINE` `...`
BY VALUE RESULT	値による呼び出しと結果： BY VALUE はサブプログラムまたはサブルーチンに渡されるパラメータに適用されますが、BY VALUE RESULT では、パラメータが値によって両方向に渡されます。つまり、実際のパラメータ値が呼び出し側のオブジェクトからサブプログラムまたはサブルーチンに渡され、呼び出し側のオブジェクトに戻るときに、実際のパラメータ値がサブプログラムまたはサブルーチンから呼び出し側オブジェクトに返されます。 BY VALUE RESULT を指定する場合は、対象フィールドのフォーマット／長さに関して、両方向のデータ転送で互換性を確保する必要があります。 **注意:** BY VALUE RESULT は、ダイアログでは使用できません。
OPTIONAL	OPTIONAL を使用しないで定義したパラメータ（デフォルト）には、呼び出し側のオブジェクトから値を渡す*必要*があります。 OPTIONAL を使用して定義したパラメータには、値を渡すことはできますが、呼び出し側のオブジェクトからこのパラメータに値を渡す必要はありません。

	呼び出し側のオブジェクトでは、スキップされるパラメータが、表記 *n*X を使用して示されます。つまり、値は渡されません。 SPECIFIED オプションを指定すると、実行時にオプションパラメータが定義されているかどうかを確認できます。
parameter-handle-definition	下記の「*パラメータハンドルの定義*」を参照してください。

パラメータハンドルの定義

parameter-handle-definition の構文：

```
handle-name   [(array-definition)] HANDLE OF { dialog-element-type }
                                              { OBJECT            }
```

構文要素の説明：

handle-name	ハンドルに割り当てる名前です。ユーザー定義変数の命名規則が適用されます。『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。
HANDLE OF *dialog-element-type*	ダイアログエレメントのタイプ。 設定可能値は、TYPE 属性の値です。 詳細については、『*ダイアログコンポーネントリファレンス*』の「*ダイアログエレメント*」および「*属性*」を参照してください。
HANDLE OF OBJECT	『*プログラミングガイド*』の「*NaturalX*」セクションで説明されているように、NaturalX と一緒に使用します。
array-definition	*array-definition*の場合、配列定義の次元の下限と上限を定義します。「*配列の次元の定義*」を参照してください。

35 アプリケーションに依存しない変数の定義

DEFINE DATA INDEPENDENT の一般的な構文：

```
DEFINE DATA
INDEPENDENT AIV-data-definition...
END-DEFINE
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

DEFINE DATA INDEPENDENT ステートメントでは、アプリケーションに依存しない変数（AIV）を定義します。

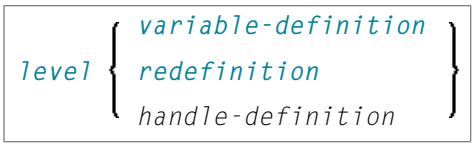
アプリケーションに依存しない変数は、その名前によって参照されます。またはこの変数の内容は、その名前を参照する 1 つのアプリケーション内で実行されるすべてのプログラミングオブジェクトによって共有されます。 変数は、この変数を参照する最初のプログラミングオブジェクトによって割り当てられ、LOGON コマンドまたは RELEASE VARIABLES ステートメントによって解除されます。

任意の INIT 節は、その節を含む、各実行プログラミングオブジェクトで評価されます（変数を割当てるプログラミングオブジェクト内だけではありません）。

構文説明

INDEPENDENT *AIV-data-definition*	DEFINE DATA INDEPENDENT ステートメントでは、1つまたは複数のアプリケーションに依存しない変数（AIV）を定義できます。 各 AIV には、下記の構文が適用されます。
END-DEFINE	Natural 予約語 END-DEFINE を使用して、DEFINE DATA ステートメントを終了させる必要があります。

AIV データ定義



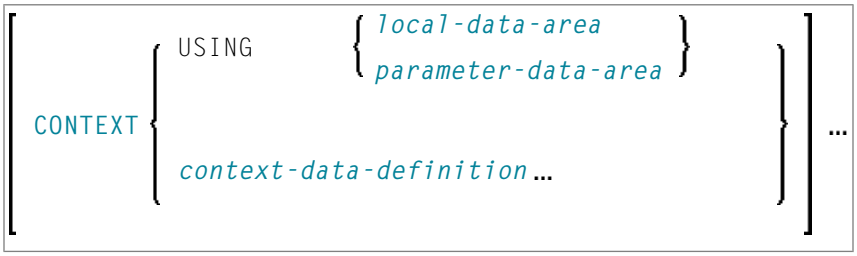
構文要素の説明：

level	アプリケーションに依存しない変数は、レベル01で定義する必要があります。他のレベルは再定義だけで使用します。
variable-definition	*variable definition* を使用して、単一の値（スカラ）または複数の値（配列）を持つ単一のフィールド／変数を定義します。「*変数定義*」を参照してください。 **注意:** アプリケーションに依存しない変数の名前は、"+" 文字で始める必要があります。
redefinition	*redefinition* を使用して、グループ、ビュー、DDM フィールド、単一のフィールド／変数（スカラまたは配列）を再定義できます。「*再定義*」を参照してください。 再定義によって生成されるフィールドは、アプリケーションに依存しない変数にすることはできません。つまり、これらのフィールドの名前を、"+"記号で始めることはできません。これらのフィールドは、ローカル変数として処理されます。
handle-definition	ハンドルは、コードのダイアログエレメントを特定し、ハンドル変数に格納されます。「*ハンドルの定義*」を参照してください。

Note: 名前の最初の文字は、"+" にする必要があります。 Natural 変数の名前に適用される規則については、『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。

36 Natural RPC 用のコンテキスト変数の定義

DEFINE DATA CONTEXT の一般的な構文：



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

DEFINE DATA CONTEXT ステートメントは、Natural リモートプロシージャコール（RPC）と一緒に使用します。このステートメントは、コンテキスト変数と呼ばれる変数の定義に使用します。コンテキスト変数では、対応する CALLNAT ステートメントで変数をパラメータとして明示的に渡さなくても、1 つの会話の中で複数のリモートサブプログラムがこの変数を使用できます。

コンテキスト変数は、その名前によって参照されます。また、この変数の内容は、その名前を参照する 1 つの会話で実行されるすべてのプログラミングオブジェクトによって共有されます。変数は、変数の定義を含むプログラミングオブジェクトが最初に実行されたときに割り当てられ、会話が終了すると解除されます。

コンテキスト変数は、非会話型 CALLNAT で使用されます。 この場合、コンテキスト変数は非会話型 CALLNAT が単独で呼び出されている間だけ存在していますが、CALLNAT から呼び出される側のすべてで共有できます。

コンテキスト変数は、会話内で呼び出されたサブプログラムで共有されることはありません。このようなサブプログラムまたはサブプログラムから呼び出される側の1つでコンテキスト変数を参照する場合は、その変数用に別のストレージエリアが割り当てられます。

任意の INIT 節は、その節を含む、各実行プログラミングオブジェクトで評価されます（変数を割当てるプログラミングオブジェクト内だけではありません）。これは、グローバル変数の INIT の動作とは異なります。

詳細については、『*Natural リモートプロシージャコール（RPC）*』ドキュメントの「*会話コンテキストの定義*」を参照してください。

制限事項

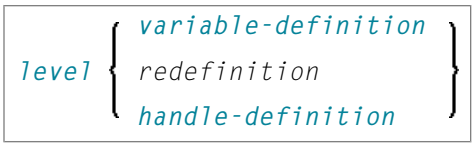
コンテキスト変数は、レベル 01 で定義する必要があります。 他のレベルは再定義だけで使用します。

構文説明

USING *local-data-area*	ローカルデータエリア（LDA）には、単一の Natural モジュールで使用されるデータ要素が含まれています。 複数のデータエリアを参照できますが、その場合は、予約語 CONTEXT と USING を繰り返す必要があります。例えば、次のようになります。 `DEFINE DATA` `  CONTEXT USING DATX_L` `  CONTEXT USING DATX_P` `  ...` `END-DEFINE ;` 詳細については、『*プログラミングガイド*』の「*別のデータエリアでのフィールドの定義*」を参照してください。
USING *parameter-data-area*	パラメータデータエリアには、サブプログラム、外部サブルーチン、またはダイアログでパラメータとして使用されるデータ要素が含まれています。
context-data-definition	コンテキストデータは、プログラムまたはルーチン内で直接定義することができます。 ダイレクトデータ定義には、下記の構文が適用されます。
END-DEFINE	Natural 予約語 END-DEFINE を使用して、DEFINE DATA ステートメントを終了させる必要があります。

コンテキストデータ定義

コンテキストデータは、プログラムまたはルーチン内で直接定義することができます。 ダイレクトデータ定義には、次の構文が適用されます。



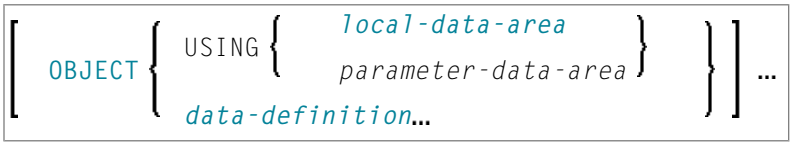
詳細については、『*プログラミングガイド*』の「*DEFINE DATA ステートメント内のフィールドの定義*」を参照してください。

level	アプリケーションに依存しない変数は、レベル 01 で定義する必要があります。他のレベルは再定義だけで使用します。
variable-definition	*variable-definition* を使用して、単一の値（スカラ）または複数の値（配列）を持つ単一のフィールド／変数を定義します。「*変数定義*」を参照してください。 **注意:** CONSTANT 節は、このコンテキストでは使用できません。
redefinition	*redefinition* を使用して、グループ、ビュー、DDM フィールド、単一のフィールド／変数（スカラまたは配列）を再定義できます。「*再定義*」を参照してください。
handle-definition	ハンドルは、コードのダイアログエレメントを特定し、ハンドル変数に格納されます。「*ハンドルの定義*」を参照してください。

Note: 再定義によって生成されるフィールドは、コンテキスト変数とはみなされません。これらのフィールドは、ローカル変数として処理されます。

37 NaturalX オブジェクトの定義

DEFINE DATA OBJECT の一般的な構文：



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

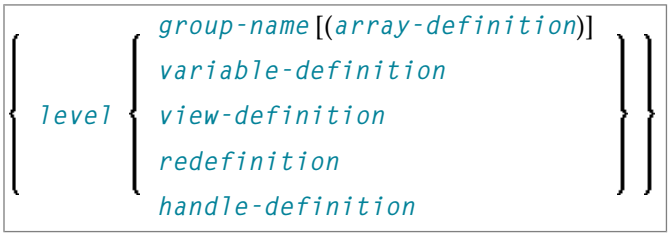
DEFINE DATA OBJECT ステートメントは、サブプログラムまたはクラス内で、NaturalX と一緒に使用します。 詳細については、『*プログラミングガイド*』の「*NaturalX*」セクションを参照してください。

構文説明

USING *local-data-area*	ローカルデータエリア（LDA）には、単一の Natural モジュールで使用されるデータ要素が含まれています。 複数のデータエリアを参照できますが、その場合は次の例のように、予約語 OBJECT および USING を繰り返す必要があります。 `DEFINE DATA` `  OBJECT USING DATX_L` `  OBJECT USING DATX_P` `  ...` `END-DEFINE ;` 詳細については、『*プログラミングガイド*』の「*別のデータエリアでのフィールドの定義*」を参照してください。
USING *parameter-data-area*	DEFINE DATA OBJECT ステートメントで定義するデータエリアは、パラメータデータエリア（PDA）でもかまいません。 PDA をオブジェクトデータエリアとして使用することにより、PDA と同じ構造を持つオブジェクトデータエリアを作成する手間を省くことができます。
data-definition	また、下記の「*ダイレクトデータ定義*」セクションで示されている構文を使用して、データを直接定義することもできます。
END-DEFINE	Natural 予約語 END-DEFINE を使用して、DEFINE DATA ステートメントを終了させる必要があります。

ダイレクトデータ定義

また、次の構文を使用してデータを直接定義することもできます。

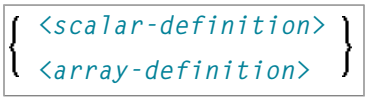


詳細については、『*プログラミングガイド*』の「*DEFINE DATA ステートメント内のフィールドの定義*」を参照してください。

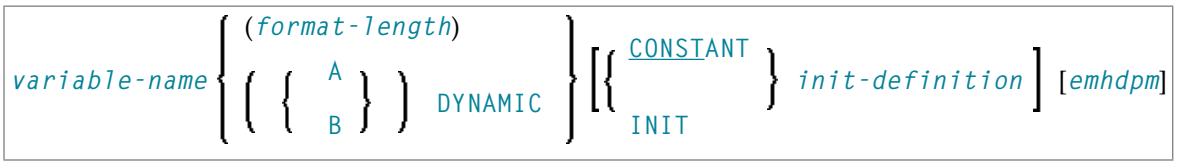
level	レベル番号は 01～99 の範囲内の 1 桁または 2 桁の数字（先頭の "0" は任意）を使用し、フィールドグルーピングと合わせて使用します。02 またはそれ以上のレベル番号が割り当てられたフィールドは、それより小さいレベル番号が割り当てられた直前のグループの一部であるとみなされます。 グループを定義すると、グループ名を使用して一連のフィールド（または 1 つのフィールドのみ）を参照できます。特定のステートメント（CALL、CALLNAT、RESET、WRITE など）の場合、グループ名をショートカットとして指定し、グループに含まれるフィールドを参照することもできます。 グループには、他のグループを含めることができます。グループにレベル番号をつけるときは、レベル番号を飛ばしてはなりません。 ビューの定義は常にレベル 1 で定義する必要があります。
group-name	グループの名前。名前は、Natural 変数名の定義ルールに準拠していなければなりません。次のセクションも参照してください。 ■ 『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」 ■ 『*プログラミングガイド*』の「*データ構造の条件指定*」
array-definition	*array-definition* の場合、配列定義の次元の下限と上限を定義します。「*配列の次元の定義*」を参照してください。
variable-definition	*variable-definition* を使用して、単一の値（スカラ）または複数の値（配列）を持つ単一のフィールド／変数を定義します。「*変数定義*」を参照してください。
view-definition	*view-definition* を使用して、データ定義モジュール（DDM）から派生したビューを定義します。「*ビューの定義*」を参照してください。
redefinition	*redefinition* を使用して、グループ、ビュー、DDM フィールド、単一のフィールド／変数（スカラまたは配列）を再定義できます。「*再定義*」を参照してください。
handle-definition	ハンドルは、コードのダイアログエレメントを特定し、ハンドル変数に格納されます。「*ハンドルの定義*」を参照してください。

38 変数定義

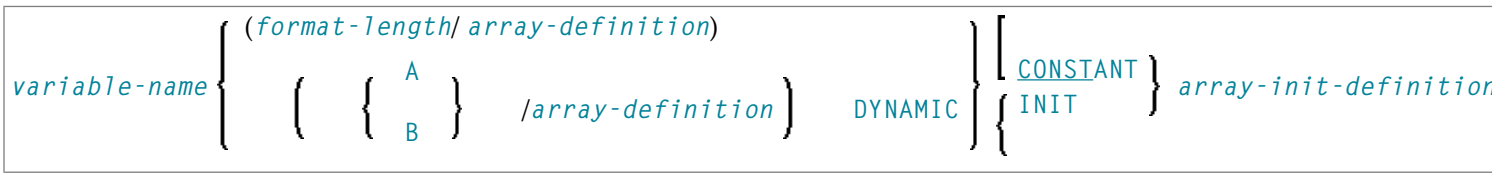
DEFINE DATA LOCAL、DEFINE DATA INDEPENDENT、DEFINE DATA CONTEXT、および DEFINE DATA OBJECT で使用する *variable-definition* では、*scalar-definition* または *array-definition* のいずれかを指定できます。



<*scalar-definition*>



<*array-definition*>



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

variable-definition を使用して、単一の値（スカラ）または複数の値（配列）を持つ単一のフィールド／変数を定義します。

構文説明

variable-name	変数に割り当てられる名前。Natural変数名のルールが適用されます。DEFINE DATA INDEPENDENT で使用する場合、変数名は "+" で始まる必要があります。 ユーザー定義変数の命名規則については、『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。

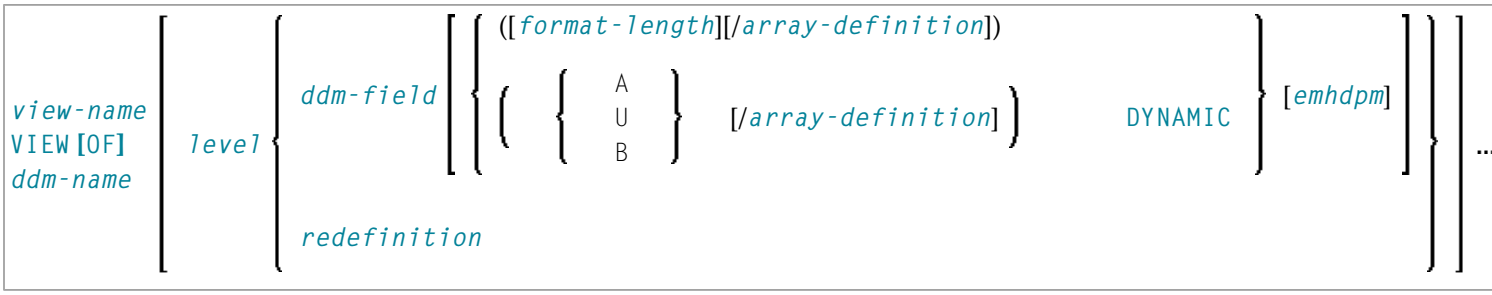
format-length	フィールドのフォーマットおよび長さ。ユーザー定義変数のフォーマットおよび長さの定義については、『*プログラミングガイド*』の「*ユーザー定義変数のフォーマットおよび長さ*」を参照してください。
A、U、または B	データタイプ：ダイナミック変数用の英数字（A）、Unicode（U）、またはバイナリ（B）。
array-definition	*array-definition* の場合、配列定義の次元の下限と上限を定義します。「*配列の次元の定義*」を参照してください。
DYNAMIC	フィールドは、DYNAMIC として定義できます。ダイナミック変数の処理については、「*ラージ変数／フィールドとダイナミック変数／フィールド*」を参照してください。
CONSTANT	変数／配列は、指定された定数として処理されます。割り当てられた定数値は、変数／配列が参照されるたびに使用されます。割り当てられた値は、プログラムの実行中には変更できません。 『*プログラミングガイド*』の「*フィールドの定義*」、「*ユーザー定義定数*」、および「*名前付き定数の定義*」も参照してください。 **注意:** 内部処理の理由から、1 つのグループ定義内で変数定義と定数定義を混在させることは許可されていません。つまり、1 つのグループに含めることができるのは、変数か定数のいずれかのみです。 CONSTANT 節を DEFINE DATA INDEPENDENT および DEFINE DATA CONTEXT で使用することはできません。 CONST 節を X-array と一緒に使用することはできません。
INIT	変数／配列に初期値が割り当てられます。 この値は、変数／配列が RESET INITIAL ステートメントで参照されるときにも使用されます。 INIT 指定をしない場合、フィールドはそのフォーマットに対応するデフォルトの初期値で初期化されます（下記の「*デフォルトの初期値*」を参照）。 『*プログラミングガイド*』の「*フィールドの定義*」の「*初期値*」も参照してください。 **注意:** DEFINE DATA INDEPENDENT および DEFINE DATA CONTEXT では、INIT 節は、その節を含む、各実行プログラミングオブジェクトで評価されます（変数を割当てるプログラミングオブジェクト内だけではありません）。これは、グローバル変数の INIT の動作とは異なります。 INIT 節を X-array と一緒に使用することはできません。
init-definition	*init-definition* オプションを使用して、変数の初期値および定数値を定義します。 「*初期値の定義*」を参照してください。
array-init-definition	*array-init-definition* を使用して、配列の初期値および定数値を定義します。 「*配列用の初期値／定数値*」を参照してください。
emhdpm	このオプションを使用して、フィールド／変数に有効な追加のパラメータを定義できます。「*フィールド／変数の **EM**、**HD**、**PM** パラメータ*」を参照してください。

デフォルトの初期値

フォーマット	デフォルトの初期値
B、F、I、N、P	0
A	空白
L	F(ALSE)
D	D' '
T	T'00:00:00'
C	(AD=D)
GUI ハンドル	NULL-HANDLE
オブジェクトハンドル	NULL-HANDLE

39 ビューの定義

DEFINE DATA LOCAL および DEFINE DATA OBJECT で使用する *view-definition* オプションの構文は、次のとおりです。



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

view-definition を使用して、データ定義モジュール（DDM）から派生したデータビューを定義します。

Note: パラメータデータエリアでは、*view-definition* を使用できません。

詳細については、『*プログラミングガイド*』の「*Adabas データベースのデータへのアクセス*」セクションで、特に次のトピックを参照してください。

- *データ定義モジュール - DDM*
- *データベース配列*
- *DEFINE DATA ビュー*

構文説明

view-name	ビューに割り当てられる名前。 Natural 変数の名前に適用される規則については、『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。
VIEW [OF] *ddm-name*	ビューを取得する元の DDM の名前。
level	レベル番号は 01～99 の範囲内の 1 桁または 2 桁の数字（先頭の "0" は任意）を使用し、フィールドグルーピングと合わせて使用します。 02 またはそれ以上のレベル番号が割り当てられたフィールドは、それより小さいレベル番号が割り当てられた直前のグループの一部であるとみなされます。

	グループを定義すると、グループ名を使用して一連のフィールド（または 1 つのフィールドのみ）を参照できます。 特定のステートメント（CALL、CALLNAT、RESET、WRITE など）の場合、グループ名をショートカットとして指定し、グループに含まれるフィールドを参照することもできます。 グループには、他のグループを含めることができます。グループにレベル番号をつけるときは、レベル番号を飛ばしてはなりません。
ddm-field	DDM から取得されるフィールドの名前。 HISTOGRAM ステートメントのビューを定義する場合、ビューには、HISTOGRAM を実行するディスクリプタのみが含まれている必要があります。
redefinition	*redefinition* を使用して、グループ、ビュー、DDM フィールド、単一のフィールド／変数（スカラまたは配列）を再定義できます。「*再定義*」セクションを参照してください。
format-length	フィールドのフォーマットおよび長さ。省略した場合、これらはDDMから取得されます。 ストラクチャードモードでは、フォーマットおよび長さの定義（提供される場合）を、DDMに指定されているフォーマットおよび長さと同じにする必要があります。 レポーティングモードでは、フォーマットおよび長さの定義（提供される場合）と、DDM に指定されているフォーマットおよび長さとの間で、タイプの互換性を確保する必要がります。
A、U、または B	データタイプ：ダイナミック変数用の英数字（A）、Unicode（U）、またはバイナリ（B）。 **注意:** 1. メインフレーム上の Adabas では、フォーマット U を LA フィールド（長さ <= 16341 バイト）に使用できますが、LB フィールド（長さ <= 1 GB）には使用できません。 2. フォーマット B は、Adabas では使用できません。
array-definition	使用されているプログラミングモードによっては、配列（ピリオディックグループフィールド、マルチプルバリューフィールド）にオカレンス数に関する情報を含める必要が生じる場合があります。 下記の「*ビューの配列定義*」を参照してください。
emhdpm	このオプションを使用して、フィールド／変数に有効な追加のパラメータを定義できます。「*フィールド／変数の EM、HD、PM パラメータ*」を参照してください。
DYNAMIC	ビューフィールドを DYNAMIC として定義します。 ダイナミック変数の処理については、「*ラージ変数／フィールドとダイナミック変数／フィールド*」セクションを参照してください。

ビューの配列定義

使用されているプログラミングモードによっては、配列（ピリオディックグループフィールド、マルチプルバリューフィールド）にオカレンス数に関する情報を含める必要が生じる場合があります。

- ストラクチャードモード
- レポーティングモード

ストラクチャードモード

配列を表すビューでフィールドを使用する場合は、次のことが適用されます。

- MU フィールドおよび PE フィールドのインデックス値を指定する必要があります。
- フォーマット／長さ指定をしない場合、値は DDM から取得されます。
- フォーマット／長さ指定をする場合は、フォーマットおよび長さをDDMのものと同じにする必要があります。

ストラクチャードモードでのデータベース固有の考慮事項：

Adabas：	MUフィールドおよびPEフィールド（DDMで定義）をビューの中で使用する場合は、これらのフィールドを配列のインデックス指定に含める必要があります。MUフィールドまたは通常のPEフィールドに対しては、例えば(1:10)のように1次元のインデックス範囲を指定します。PE グループの中にある MU フィールドに対しては、例えば (1:10,1:5) のように 2 次元のインデックス範囲を指定します。		
Tamino：	**DDM 定義**	**指定可**	**指定不可**
	`A(*:X2)`	`A(*:Y2) Y2=<X2` `A(Y1:Y2) Y2>Y1` `Y2=<X2 A(Z:Z+Y) Y>=0`	`A(*:*)` `A(Y1:*)`
	`A(X1:*)`	`A(Y1:*) Y1>=X1` `A(Y1:Y2) Y2>=X1, Y1>=X1` `A(Z:Z+Y) Y>=0`	`A(*:*)` `A(*:Y2)`
	`A(X1:X2)`	`A(Y1:Y2) Y2<Y1` `A(Z:Z+Y) 0=<Y>=X2-X1+1`	`A(*:*)` `A(Y1:*)` `A(*:Y2)`

ストラクチャードモードの例：

```
DEFINE DATA LOCAL
1 EMP1 VIEW OF EMPLOYEES
  2 NAME(A20)
  2 ADDRESS-LINE(A20 / 1:2)

1 EMP2 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(1:2)

1 EMP3 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(2)

1 #K (I4)
1 EMP4 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(#K:#K+1)
END-DEFINE
END
```

レポーティングモード

このモードでは、ストラクチャードモードと同じ規則が有効です。ただし、2つの例外があります。

- インデックス値を提供する必要はありません。この場合、不明な次元のインデックス範囲は(1:1)に設定されます。
- フォーマット／長さ指定は、DDMでの指定と異なる可能性があります。その場合は、フォーマットおよび長さの定義と、DDMに指定されているフォーマットおよび長さとの間で、タイプの互換性を確保する必要があります。

例：

```
DEFINE DATA LOCAL
1 EMP1 VIEW OF EMPLOYEES
  2 NAME(A30)
  2 ADDRESS-LINE(A35 / 5:10)

1 EMP2 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(A40)        /* ADDRESS LINE (1:1) IS ASSUMED

1 EMP3 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE             /* ADDRESS LINE (1:1) IS ASSUMED

1 #K (I4)
1 EMP4 VIEW OF EMPLOYEES
```

```
  2 NAME
  2 ADDRESS-LINE(#K:#K+1)
END-DEFINE
END
```

40 再定義

DEFINE DATA LOCAL、DEFINE DATA PARAMETER、DEFINE DATA INDEPENDENT、DEFINE DATA CONTEXT、および DEFINE DATA OBJECT で使用する *redefinition* オプションの構文は、次のとおりです。

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

*redefinition*を使用して、グループ、ビュー、DDMフィールド、単一のフィールド／変数（スカラまたは配列）を再定義できます。

Notes:

1. ビューまたはDDMフィールドの「再定義」は、*parameter-data-definition*には適用されません。
2. Unicodeフィールドを、英数字（A）フィールドまたは数値（N）フィールドとして再定義することはできません。

『*プログラミングガイド*』の「*フィールドの再定義*」も参照してください。

制限事項

- ハンドル、X-array、ダイナミック変数を再定義すること、および再定義節に含めることはできません。
- ハンドル、X-array、またはダイナミック変数を含むグループの再定義のみを行うことはできますが、問題の要素を含めることや超えることはできません。

構文説明

field-name	再定義するグループ、ビュー、DDM フィールド、または単一フィールドの名前。
level	レベル番号は 01～99 の範囲内の 1 桁または 2 桁の数字（先頭の "0" は任意）を使用し、フィールドグルーピングと合わせて使用します。 02 またはそれ以上のレベル番号が割り当てられたフィールドは、それより小さいレベル番号が割り当てられた直前のグループの一部であるとみなされます。
rgroup	再定義によって生成されるグループの名前。 **注意:** *view-definition* 内の *redefinition* では、*rgroup* の名前は、基礎となる DDM 内のフィールド名とは異なっている必要があります。
rfield	再定義によって生成されるフィールドの名前。 **注意:** *view-definition* 内の *redefinition* では、*rfield* の名前は、基礎となる DDM 内のフィールド名とは異なっている必要があります。
format-length	生成されるフィールド（*rfield*）のフォーマットおよび長さ。
array-definition	*array-definition* の場合、配列定義の次元の下限と上限を定義します。「*配列の次元の定義*」を参照してください。
FILLER *n*X	この表記では、再定義するフィールドに *n* フィラーバイト（使用しないセグメント）を定義します。 末尾の充填バイトは任意指定です。

REDEFINE の使用例

例 1：

```
   DEFINE DATA LOCAL
     01 #VAR1 (A15)
     01 #VAR2
        02 #VAR2A (N4.1)
INIT <0>
        02 #VAR2B (P6.2)
INIT <0>
     01 REDEFINE #VAR2
        02 #VAR2RD (A10)
   END-DEFINE
   ...
```

例 2：

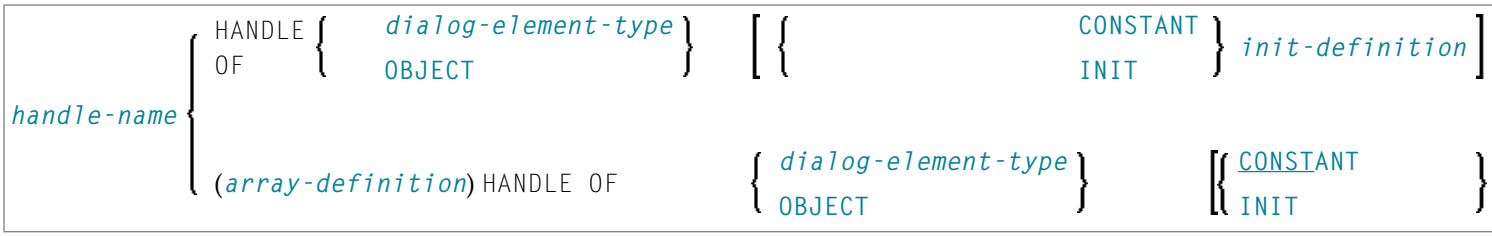
```
  DEFINE DATA LOCAL
  01 MYVIEW VIEW OF STAFF
     02 NAME
     02 BIRTH
     02 REDEFINE BIRTH
       03 BIRTH-YEAR  (N4)

        03 BIRTH-MONTH (N2)
        03 BIRTH-DAY   (N2)
  END-DEFINE
  ...
```

例 3：

```
  DEFINE DATA LOCAL

  1 #FIELD (A12)
  1 REDEFINE #FIELD
    2 #RFIELD1 (A2)
    2 FILLER 2X
    2 #RFIELD2 (A2)
    2 FILLER 4X
    2 #RFIELD3 (A2)
  END-DEFINE
  ...
```

41 ハンドルの定義

DEFINE DATA LOCAL、DEFINE DATA OBJECT、DEFINE DATA PARAMETER、DEFINE DATA INDEPENDENT、および DEFINE DATA CONTEXT で使用する *handle-definition* の構文は、次のとおりです。



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

ハンドルは、コードのダイアログエレメントを特定し、ハンドル変数に格納されます。 詳細については、『*プログラミングガイド*』の「*NaturalX*」セクションを参照してください。

DEFINE DATA ステートメントでの HANDLE 定義は、ダイアログエレメントまたはダイアログを作成すると、自動的に生成されます。

ハンドルを定義すると、任意のステートメントでハンドル名を使用して、定義したダイアログエレメントタイプの属性値をクエリ、設定、または変更することができます。

『*プログラミングガイド*』の「*イベントドリブンプログラミング手法*」も参照してください。

handle-definition の例：

```
1 #SAVEAS-MENUITEM HANDLE OF MENUITEM
1 #OK-BUTTON (1:10) HANDLE OF PUSHBUTTON
```

構文説明

handle-name	ハンドルに割り当てる名前です。ユーザー定義変数の命名規則が適用されます。『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。
HANDLE OF *dialog-element-type*	ダイアログエレメントのタイプ。 設定可能値は、TYPE 属性の値です。 詳細については、『*ダイアログコンポーネントリファレンス*』の「*ダイアログエレメント*」および「*属性*」を参照してください。
HANDLE OF OBJECT	『*プログラミングガイド*』の「*NaturalX*」セクションで説明されているように、NaturalX と一緒に使用します。
CONSTANT	変数／配列は、指定された定数として処理されます。割り当てられた定数値は、変数／配列が参照されるたびに使用されます。割り当てられた値は、プログラムの実行中には変更できません。 **注意:** 1. 内部処理の理由から、1 つのグループ定義内で変数定義と定数定義を混在させることは許可されていません。つまり、1 つのグループに含めることができるのは、変数か定数のいずれかのみです。 2. CONSTANT 節を DEFINE DATA INDEPENDENT および DEFINE DATA CONTEXT で使用することはできません。
INIT	変数／配列に初期値が割り当てられます。 この値は、変数／配列が RESET INITIAL ステートメントで参照されるときにも使用されます。 **注意:** DEFINE DATA INDEPENDENT および DEFINE DATA CONTEXT では、INIT 節は、その節を含む、各実行プログラミングオブジェクトで評価されます（変数を割当てるプログラミングオブジェクト内だけではありません）。これは、グローバル変数の INIT の動作とは異なります。
init-definition	*init-definition* オプションを使用して、変数の初期値および定数値を定義します。 「*初期値の定義*」を参照してください。
array-definition	*array-definition* の場合、配列定義の次元の下限と上限を定義します。「*配列の次元の定義*」を参照してください。
array-init-definition	配列に初期値が割り当てられます。 この値は、配列が RESET INITIAL ステートメントで参照されるときにも使用されます。

42 配列の次元の定義

array-dimension-definition は次のステートメントで使用します。DEFINE DATA OBJECT、および DEFINE DATA LOCAL、DEFINE DATA INDEPENDENT、DEFINE DATA CONTEXT、DEFINE DATA OBJECT の *variable-definition* オプション。また、DEFINE FUNCTION ステートメントでも使用します。

array-dimension-definition の構文は、次のとおりです。

{*bound*[:*bound*]},... 3

このchapterでは、次のトピックについて説明します。

機能

array-dimension-definition の場合、配列定義の次元の下限と上限を定義します。

最大 3 次元までの配列を定義することができます。

『*プログラミングガイド*』の「*配列*」も参照してください。

構文説明

bound	次のいずれかが境界になります。 ■ 整数定数 ■ 事前に定義された名前付き定数 ■ （データベース配列用に）事前に定義されたユーザー定義変数 ■ "*" で拡張可能な境界を定義します。それ以外の場合は、X-array として認識されます。 1 つの境界だけを指定した場合、その値は上限を表し、下限は 1 であると想定されます。

X-Array

配列の少なくとも 1 つの次元で少なくとも 1 つの境界を拡張可能に指定すると、その配列は X-array（eXtensible array）と呼ばれます。1つの次元で拡張可能なのは1つの境界だけであり（上限または下限のいずれか）、両方を拡張可能にすることはできません。 多次元配列では、例えば `#a(1:100, 1:*)` のように、定数と拡張可能な境界が混在していることがあります。

例：

```
DEFINE DATA LOCAL
1 #ARRAY1(I4/1:10)
1 #ARRAY2(I4/10)
1 #X-ARRAY3(I4/1:*)
1 #X-ARRAY4(I4/*,1:5)
1 #X-ARRAY5(I4/*:10)
1 #X-ARRAY6(I4/1:10,100:*,*:1000)
END-DEFINE
```

次の表に、上記のプログラムで使用する配列の境界をわかりやすく示します。

	次元 1		次元 2		次元 3	
	下限	上限	下限	上限	下限	上限
#ARRAY1	1	10	-	-	-	-
#ARRAY2	1	10	-	-	-	-
#X-ARRAY3	1	拡張可能	-	-	-	-
#X-ARRAY4	1	拡張可能	1	5	-	-
#X-ARRAY5	拡張可能	10	-	-	-	-
#X-ARRAY6	1	10	100	拡張可能	拡張可能	1000

配列定義の例：

```
#ARRAY2(I4/10)                 /* a one-dimensional array with 10 occurrences (1:10)
#X-ARRAY4(I4/*,1:5)               /* a two-dimensional array
#X-ARRAY6(I4/1:10,100:*,*:1000)  /* a three-dimensional array
```

パラメータデータエリア内の可変配列

パラメータデータエリアには、可変オカレンス数を含む配列を指定できます。 この指定は、インデックス表記 `1:V` を使用して行います。

例 1: `#ARR01 (A5/1:V)`

例 2: `#ARR02 (I2/1:V,1:V)`

変数インデックス表記 `1:V` を含むパラメータ配列は、次の長さでのみ再定義できます。

- 例えば、次のように `1:V` インデックスが右端の場合は、そのエレメンタリフィールドの長さになります。

  `#ARR(A6/1:V)` は、最大 6 バイトまでの長さで再定義できます。
  `#ARR(A6/1:2,1:V)` は、最大 6 バイトまでの長さで再定義できます。

#ARR(A6/1:2,1:3,1:V) は、最大 6 バイトまでの長さで再定義できます。

- 右端の固定オカレンス数とエレメンタリフィールドの長さの積は、例えば次のようになります。

#ARR(A6/1:V,1:2) は、最大 2 x 6 = 12 バイトまでの長さで再定義できます。
#ARR(A6/1:V,1:3,1:2) は、最大 3 x 2 x 6 = 36 バイトまでの長さで再定義できます。
#ARR(A6/1:2,1:V,1:3) は、最大 3 x 6 = 18 バイトまでの長さで再定義できます。

変数インデックス表記 1:V は、再定義の中では使用できません。

例：

```
DEFINE DATA PARAMETER
 1 #ARR(A6/1:V)
 1 REDEFINE #ARR
   2 #R-ARR(A1/1:V)   /* (1:V) is not allowed in a REDEFINE block
END-DEFINE
```

オカレンス数はコンパイル時には不明であるため、ステートメント INPUT、WRITE、READ WORK FILE、および WRITE WORK FILE でインデックス表記 (*) を使用してオカレンス数を参照することはできません。インデックス表記 (*) は、すべての次元に適用されるか、またはどの次元にも適用されないかのいずれかになります。

有効な例：

```
#ARR01 (*)
#ARR02 (*,*)
#ARR01 (1)
#ARR02 (5,#FIELDX)
#ARR02 (1,1:3)
```

無効な例：

```
#ARRAYY (1,*)  /* not allowed
```

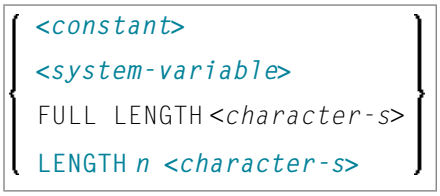
ラインタイムエラーを回避するために、このような配列の最大オカレンス数は、別のパラメータを経由してサブプログラムまたはサブルーチンに渡す必要があります。 代わりに、システム変数 *OCCURRENCE を使用することもできます。

Notes:

1. インデックス "1:V" を含むパラメータデータエリアを使用する場合（つまり、DEFINE DATA LOCAL ステートメントで指定した場合）は、"V" という名前の変数を CONSTANT として定義する必要があります。
2. ダイアログでは、インデックス "1:V" を BY VALUE と一緒に使用することはできません。

43 初期値の定義

DEFINE DATA LOCAL、DEFINE DATA INDEPENDENT、DEFINE DATA CONTEXT、および DEFINE DATA OBJECT の *variable-definition* オプションで使用する *init-definition* の構文は、次のとおりです。



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

init-definition オプションを使用して、変数の初期値および定数値を定義します。

Note: *variable-definition* オプションで、キーワード INIT が初期化に使用されている場合、変数の内容に影響するステートメントによって値を変更することができます。キーワード CONST が初期化に使用されている場合、値を変更しようとすると、コンパイラで拒否されます。

『*プログラミングガイド*』の「*フィールドの定義*」の「*初期値*」も参照してください。

制限事項

再定義されたフィールドでは、*init-definition* は許可されていません。

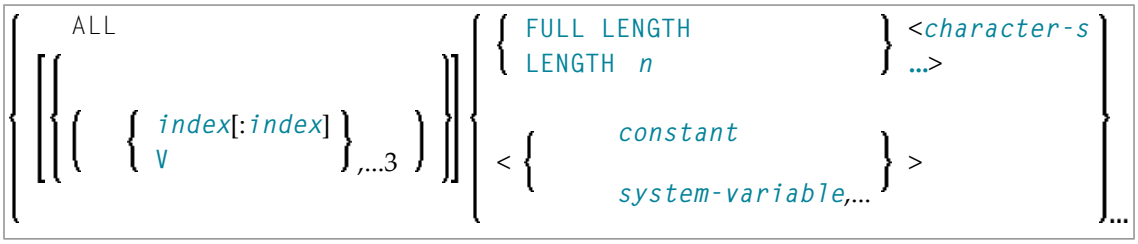
構文説明

<constant>	変数の初期化に使用する定数値、またはフィールドに割り当てる定数値です。 定数の詳細については、『*プログラミングガイド*』の「*ユーザ定義定数*」を参照してください。

<system-variable>	変数の初期値として Natural システム変数の値を指定することもできます。 例： `DEFINE DATA LOCAL` `1 #MYDATE (D) INIT <*DATX>` `END-DEFINE` **注意:** `RESET INITIAL` ステートメント内でシステム変数が参照されると、システム変数は再び評価されます。つまり、プログラムの実行を開始したときにシステム変数に含まれていた値にリセットされるのではなく、`RESET INITIAL` ステートメントを実行するときに含まれている値にリセットされます。
FULL LENGTH *<character-s>* **LENGTH** *n* *<character-s>*	初期値として、特定の 1 文字または文字列を、変数の全体または一部に挿入します。これは、英数字変数（コードページまたは Unicode）変数だけに有効です。 `FULL LENGTH` オプションを使用すると、指定した *character* または *characters* がフィールド全体に挿入されます。 次の例では、フィールド全体にアスタリスクが挿入されます。 `DEFINE DATA LOCAL` `1 #FIELD (A25) INIT FULL LENGTH <'*'>` `END-DEFINE` `LENGTH` *n* オプションを使用すると、フィールドの先頭 *n* 桁に、指定した *character* または *characters* が挿入されます。 *n* は数値定数にする必要があります。 次の例では、フィールドの先頭 4 桁に感嘆符が挿入されます。 `DEFINE DATA LOCAL` `1 #FIELD (A25) INIT LENGTH 4 <'!'>` `END-DEFINE`

44 配列用の初期値／定数値

DEFINE DATA LOCAL、DEFINE DATA INDEPENDENT、DEFINE DATA CONTEXT、および DEFINE DATA OBJECT の *variable-definition* オプションで使用する *array-init-definition* オプションの構文は、次のとおりです。



このchapterでは、次のトピックについて説明します。

機能

array-init-definition を使用して、配列の初期値および定数値を定義します。

Note: *variable-definition* オプションで、キーワード INIT が初期化に使用されている場合、変数の内容に影響するステートメントによって値を変更することができます。キーワード CONST が初期化に使用されている場合、値を変更しようとすると、コンパイラで拒否されます。

『プログラミングガイド』の「*フィールドの定義*」の「*初期値*」、および特に次のセクションも参照してください。

- *初期値*
- *ユーザー定義定数*

制限事項

再定義されたフィールドでは、*array-init-definition* は許可されていません。

構文説明

ALL	配列のすべての次元にあるすべてのオカレンスを、同じ値で初期化します。
index	*index* で指定した配列オカレンスのみを初期化します。 *index* を指定する場合、*constant* で指定できる値は1つだけです。つまり、指定したオカレンスはすべて、同じ値で初期化されます。
V	1つの次元のオカレンスを異なる値で初期化する場合、この表記は多次元配列にのみ関係しています。"V" は、"V" で指定した次元のすべてのオカレンスで構成されるインデックス範囲を示します。つまり、その次元のオカレンスは、すべて初期化されます。 配列ごとに 1 つの次元のみを、"V" で指定することができます。 オカレンスは、オカレンスごとにその次元用に指定した値で初期化されます。 値の数は、"V" で指定した次元のオカレンス数を超えないようにする必要があります。
constant	配列を初期化に使用する定数（値）（INIT）、または配列に割り当てる定数（CONSTANT）です。定数の詳細については、『*プログラミングガイド*』の「*ユーザ定義定数*」を参照してください。 **注意:** 値の指定がないオカレンスは、デフォルト値で初期化されます。
system-variable	配列の初期値として Natural システム変数の値を指定することもできます。 **注意:** 複数の定数値またはシステム変数を指定する場合は、それぞれを、INPUT 区切り文字（ID セッションパラメータで指定した文字）またはコンマ（,）で区切る必要があります。セッションパラメータ DC でコンマを小数点文字として定義している場合は、この目的でコンマを使用することはできません。
FULL LENGTH **LENGTH** *n*	初期値として、特定の 1 文字または文字列を、変数の全体または一部に挿入します（フォーマット A または U の変数にのみ有効）。 FULL LENGTH オプションを使用すると、指定した *character* または *characters* が配列オカレンス全体に挿入されます。 LENGTH *n* オプションを使用すると、配列オカレンスの先頭 *n* 桁に、指定した *character* または *characters* が挿入されます。 FULL LENGTH または LENGTH *n* では、*system-variable* を指定することはできません。 1 つの *array-init-definition* の中では、FULL LENGTH または LENGTH *n* のいずれか一方だけを指定できます。両方の表記を混在させることはできません。

配列に対する **LENGTH** *n* の例：

この例では、配列の各オカレンスの先頭 5 桁に NONON が挿入されます。

```
DEFINE DATA LOCAL
1 #FIELD (A25/1:3) INIT ALL LENGTH 5 <'NO'>
...
END-DEFINE
```

配列への初期値の割り当てについては、多数の例が『*プログラミングガイド*』に記載されています。

45 フィールド／変数の EM、HD、PM パラメータ

DEFINE DATA LOCAL と DEFINE DATA OBJECT の *view-definition* オプション、DEFINE DATA LOCAL、DEFINE DATA INDEPENDENT、DEFINE DATA CONTEXT、および DEFINE DATA OBJECT の *variable-definition* オプションで使用する *emhdpm* オプションの構文は、次のとおりです。

```
( [ EM=value  ] [HD='text'] [PM=value])
  [ EMU=value ]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

このオプションを使用して、フィールド／変数に有効な追加のパラメータを定義できます。

Note: 指定したデータベースフィールドが編集マスク（EM=）またはヘッダー（HD=）のいずれでもない場合は、**DDM** で指定されているデフォルト編集マスクとデフォルトヘッダーが使用されます。 ただし、デフォルト編集マスクとデフォルトヘッダーのいずれか 1 つを指定した場合は、DDM からのもう 1 つのデフォルトは使用*されません*。

構文説明

EM=*value*	このパラメータは、I/O ステートメントでフィールドを表示する際に使用される編集マスクを定義するために使用します。 『*パラメータリファレンス*』でセッションパラメータ EM を参照してください。
EMU=*value*	このパラメータでは、I/O ステートメントでフィールドを表示する際に使用される Unicode 編集マスクを定義します。 『*パラメータリファレンス*』でセッションパラメータ EMU を参照してください。
HD='*text***'**	このパラメータでは、フィールドのデフォルトヘッダーとして使用されるヘッダーを定義します。
PM=*value*	このパラメータでは、フィールドの出力方法を示す出力モードを設定します。 『*パラメータリファレンス*』でセッションパラメータ PM を参照してください。

46 DEFINE DATA ステートメントの使用例

次のトピックについて説明します。

例 1 - DEFINE DATA LOCAL（ダイレクトデータ定義）

```
** Example 'DDAEX1': DEFINE DATA
************************************************************************
DEFINE DATA LOCAL
1 #VAR1     (A15)
1 #VAR2
  2 #VAR2A  (N4.1) INIT <1111>
  2 #VAR2B  (N6.2) INIT <222222>
1 REDEFINE #VAR2
  2 #VAR2C  (A2)
  2 #VAR2D  (A2)
  2 #VAR2E  (A6)
END-DEFINE
*
WRITE NOTITLE '=' #VAR2A / '=' #VAR2B /
              '=' #VAR2C / '=' #VAR2D / '=' #VAR2E
*
END
```

プログラム **DDAEX1** の出力：

```
#VAR2A:  1111.0
#VAR2B:  222222.00
#VAR2C: 11
#VAR2D: 11
#VAR2E: 022222
```

例 2 - DEFINE DATA LOCAL（配列の定義／初期化）

```
** Example 'DDAEX2': DEFINE DATA (array definition/initialization)
************************************************************************
DEFINE DATA LOCAL
1 #VAR1 (A1/1:2,1:2) INIT (1,V)   <'A','B'>
1 #VAR2 (N5/1:2,1:3) INIT (1,2)   <200>
1 #VAR3 (A1/1:4,1:3) INIT (V,2:3) <'W','X','Y','Z'>
END-DEFINE
*
WRITE NOTITLE '=' #VAR1 (1,1) '=' #VAR1 (1,2)
            / '=' #VAR1 (2,1) '=' #VAR1 (2,2)
```

```
*
WRITE     /// '=' #VAR2 (1,1) '=' #VAR2 (1,2)
            / '=' #VAR2 (2,1) '=' #VAR2 (2,2)
*
WRITE     /// '=' #VAR3 (1,1) '=' #VAR3 (1,2) '=' #VAR3 (1,3)
WRITE       / '=' #VAR3 (2,1) '=' #VAR3 (2,2) '=' #VAR3 (2,3)
WRITE       / '=' #VAR3 (3,1) '=' #VAR3 (3,2) '=' #VAR3 (3,3)
WRITE       / '=' #VAR3 (4,1) '=' #VAR3 (4,2) '=' #VAR3 (4,3)
*
END
```

プログラム **DDAEX2** の出力：

```
#VAR1: A #VAR1: B
#VAR1:   #VAR1:



#VAR2:       0 #VAR2:    200
#VAR2:       0 #VAR2:      0



#VAR3:   #VAR3: W #VAR3: W

#VAR3:   #VAR3: X #VAR3: X

#VAR3:   #VAR3: Y #VAR3: Y

#VAR3:   #VAR3: Z #VAR3: Z
```

例 3 - DEFINE DATA（ビューの定義、配列の再定義）

```
** Example 'DDAEX3': DEFINE DATA (view definition, array redefinition)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
*
1 #ARRAY    (A75/1:4)
1 REDEFINE #ARRAY
  2 #ALINE  (A25/1:4,1:3)
1 #X        (N2) INIT <1>
1 #Y        (N2) INIT <1>
```

```
END-DEFINE
*
FORMAT PS=20
LIMIT 5
FIND EMPLOY-VIEW WITH NAME = 'SMITH'
  MOVE NAME            TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE           TO #ALINE (#X+3,#Y)
  IF  #Y = 3
    RESET INITIAL #Y
    PERFORM PRINT
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
    PERFORM PRINT
  END-ENDDATA
END-FIND
*
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
END-SUBROUTINE
*
END
```

プログラム **DDAEX3** の出力：

```

SMITH                     SMITH                     SMITH
ENGLANDSVEJ 222           3152 SHETLAND ROAD        14100 ESWORTHY RD.
                          MILWAUKEE                 MONTERREY
554349                    877-4563                  994-2260

SMITH                     SMITH
5 HAWTHORN                13002 NEW ARDEN COUR
```

```
OAK BROOK                SILVER SPRING
150-9351                 639-8963
```

例4 - DEFINE DATA（グローバル、パラメータ、およびローカルデータエリア）

```
** Example 'DDAEX4': DEFINE DATA (global and local data area definition)
************************************************************************
DEFINE DATA
GLOBAL
  USING DDAEX4G
LOCAL
1 #FIELD1 (A10)
1 #FIELD2 (N5)
END-DEFINE
*
MOVE 'HELLO' TO #FIELD1
MOVE 123     TO #FIELD2
*
CALLNAT 'DDAEX4N' #FIELD1 #FIELD2
*
END
```

プログラム **DDAEX4** によって使用されるグローバルデータエリア **DDAEX4G**：

```
1 GLOBAL-FIELD                     A   10
```

プログラム **DDAEX4** によって呼び出されるサブプログラム **DDAEX4N**：

```
** Example 'DDAEX4N': DEFINE DATA PARAMETER (called by DDAEX4)
************************************************************************
DEFINE DATA
PARAMETER
1 #FIELDA (A10)
1 #FIELDB (N5)
END-DEFINE
*
WRITE '=' #FIELDA '=' #FIELDB
END
```

プログラム **DDAEX4** の出力：

```
Page      1                                                  05-01-12  08:55:53

#FIELDA: HELLO      #FIELDB:    123
```

例 5 - DEFINE DATA（初期化）

```
** Example 'DDAEX5': DEFINE DATA (initialization)
************************************************************************
DEFINE DATA LOCAL
1 #START-DATE (D)   INIT <*DATX>
1 #UNDERLINE  (A50) INIT FULL LENGTH <'_'>
1 #SCALE      (A65) INIT LENGTH 65 <'....+..../'>
END-DEFINE
*
WRITE NOTITLE #START-DATE (DF=L)
            / #UNDERLINE
            / #SCALE
END
```

プログラム **DDAEX5** の出力：

```
2005-01-12
_________________________________________________
....+..../....+..../....+..../....+....+....+....+....+....+....+....+....
```

例 6 - DEFINE DATA（可変配列）

```
** Example 'DDAEX6': DEFINE DATA (variable array with (1:V))
************************************************************************
DEFINE DATA LOCAL
1 #ARRAY   (A1/1:10)
1 #MAX-ARR (P3)
END-DEFINE
*
#ARRAY (1) := 'R'
#ARRAY (2) := 'E'
#ARRAY (3) := 'D'
#MAX-ARR   := 4
*
WRITE #ARRAY(*)
*
```

```
CALLNAT 'DDAEX6N' #ARRAY(1:4) #MAX-ARR
*
WRITE #ARRAY(*)
*
*
#MAX-ARR   := 5
*
CALLNAT 'DDAEX6N' #ARRAY(1:5) #MAX-ARR
*
WRITE #ARRAY(*)
*
END
```

プログラム **DDAEX6** によって呼び出されるサブプログラム **DDAEX6N**：

```
** Example 'DDAEX6N': DEFINE DATA (variable array with (1:V))
************************************************************************
DEFINE DATA
PARAMETER
1 #STRING (A1/1:V)
1 #MAX    (P3)
END-DEFINE
*
IF #MAX = 4
  MOVE 'B' TO #STRING (1)
  MOVE 'L' TO #STRING (2)
  MOVE 'U' TO #STRING (3)
  MOVE 'E' TO #STRING (4)
END-IF
*
IF #MAX = 5
  MOVE 'W' TO #STRING (1)
  MOVE 'H' TO #STRING (2)
  MOVE 'I' TO #STRING (3)
  MOVE 'T' TO #STRING (4)
  MOVE 'E' TO #STRING (5)
END-IF
END
```

プログラム **DDAEX4** の出力：

```
Page      1                                                 05-01-12  09:06:43

R E D
B L U E
W H I T E
```

47 DEFINE FUNCTION

```
DEFINE FUNCTION function-name
[return-data-definition]
[function-data-definition]
statement...
END-FUNCTION
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連トピック：*ユーザー定義関数*

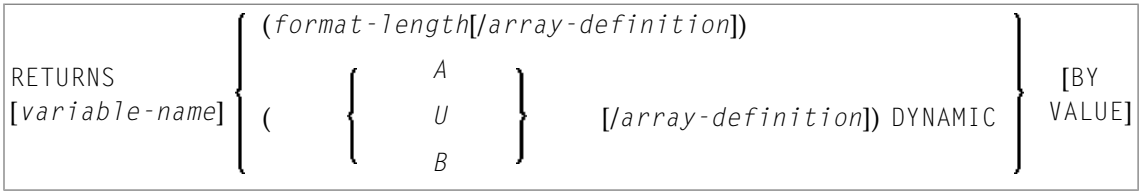
機能

DEFINE FUNCTION ステートメントでは、新規ユーザー定義関数（Natural ステートメントのオペランドの代わりに呼び出し可能）を作成します。 ファンクションは、オブジェクトタイプ FUNCTION の中だけで定義できます。このオブジェクトタイプの詳細については、『*プログラミングガイド*』の「ファンクション」を参照してください。

構文説明

function-name	*function-name*は、定義される Natural ファンクションの記号名です。この名前は、ユーザー定義変数と同じ命名規則に適合している必要があります。詳細については、『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。 したがって、名前は最大 32 文字で構成され、先頭の文字は英字または "#" などの一部の特殊文字になります。 1つのライブラリ（STEPLIB メカニズムのライブラリを含む）に同じファンクション名を 2 回使用することはできません。 ファンクションの多重定義は許可されていません。 したがって、すべてのファンクション定義の名前は一意である必要があります。
return-data-definition	この節の詳細については、下記の「*戻りデータ定義*」を参照してください。
function-data-definition	この節の詳細については、下記の「*ファンクションデータ定義*」を参照してください。
END-FUNCTION	DEFINE FUNCTION ステートメントを終了するには、Natural 予約語 END-FUNCTION を使用する必要があります。

戻りデータ定義

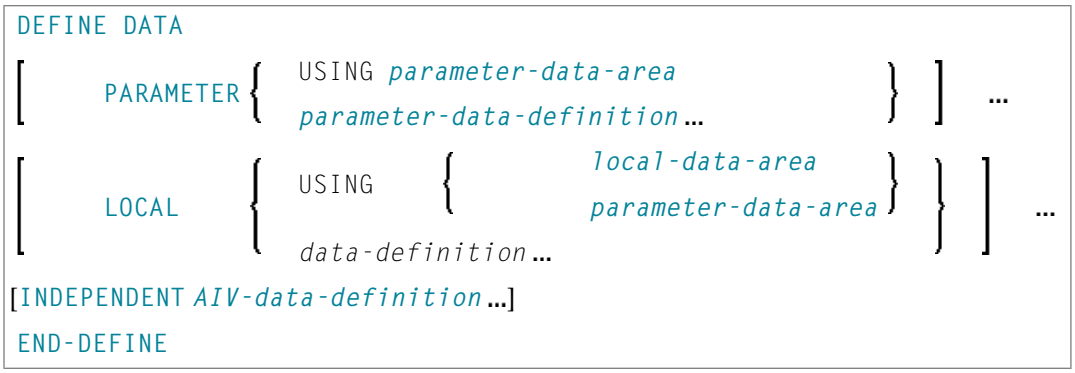


構文要素の説明：

RETURNS	各ファンクションには、戻り変数の定義を 1 つだけ含めることができます。つまり、RETURNS 節を 1 つだけ指定できます。
variable-name	戻り値は *variable-name* を使用して割り当てることができます。定義内で明示的な変数名を指定していない場合、ファンクション名が戻り変数として使用されます。 戻り値を配列にすることはできません。
BY VALUE	BY VALUE RESULT または by reference（参照渡し）として各パラメータを定義すると、各パラメータを使用して呼び出し元に値を戻すことができます。 再帰的なファンクションコールをファンクション定義内で使用できます。 RETURNS 節の中で BY VALUE キーワードを使用している場合、ファンクションの戻り値は、RETURNS 節で設定される戻りの *format-length* に変換されます。
format-length	BY VALUE を使用しない場合、RETURNS 節の *format-length* は、実行時に評価されるファンクションによって返される *format-length* と一致している必要があります。
array-definition	*array-definition* では、配列定義の次元の下限と上限を定義します。 詳細については、DEFINE DATA ステートメントの説明および「*配列の次元の定義*」を参照してください。
DYNAMIC	パラメータは DYNAMIC として定義できます。ダイナミック変数の処理については、「*ラージ変数／フィールドとダイナミック変数／フィールド*」を参照してください。

ファンクションデータ定義

各ファンクションオブジェクトには、ファンクションデータ定義を1つだけ含めることができます。



ファンクションでは、グローバルデータエリアを使用する別のNaturalオブジェクトを呼び出すときに、独自のグローバルデータエリア（GDA）を確立します。 したがって、呼び出し側オブジェクトの現在のグローバルデータエリアを変更することはできません。また、GDAをファンクション内で定義することはできません。

例

ファンクション定義を含んでいるファンクションオブジェクト：

```
DEFINE FUNCTION GET-FIRST-BYTE
  RETURNS (A1)
  DEFINE DATA PARAMETER
  1 #PARA (A10)
  END-DEFINE
  GET-FIRST-BYTE := #PARA /* return value is assigned
END-FUNCTION
END
```

48 DEFINE PRINTER

```
DEFINE PRINTER ([logical-printer-name=]n)
 [OUTPUT operand1]
 [PROFILE operand2]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

DEFINE PRINTER ステートメントは、記号名をレポート番号に割り当て、論理出力先へのレポートの割り当てを制御するために使用します。 これにより、さまざまな論理印刷キューへの出力をより柔軟に作成できます。

このステートメントの実行時に、指定したプリンタがすでに開いていた場合、プリンタはステートメントにより暗黙的に閉じられます。 ただし、プリンタを明示的に閉じる場合は、CLOSE PRINTER ステートメントを使用する必要があります。

構文説明

オペランド定義テーブル：

オペランド	構文要素			フォーマット	オペランド参照	ダイナミック定義
operand1	C	S		A	可	不可
operand2	C	S		A	可	不可

構文要素の説明：

(*n*)	プリンタ番号（レポート番号）： レポート番号 *n* は、0～31 の範囲にある値になります。 この番号は、DISPLAY/WRITE または CLOSE PRINTER ステートメントでも使用されます。 レポート番号 0 は、メインレポートの出力チャネルを示します。 PRINT、WRITE、または DISPLAY などの出力ステートメントのみが影響を受けます。 INPUT ステートメントは影響を受けません。
logical-printer-name	論理プリンタ名： オプションで、論理名 *logical-printer-name* をプリンタ *n* に割り当てることができます。 この名前は、DISPLAY/WRITE ステートメントで *rep* 表記に使用できます。 *logical-printer-name* の命名規則はユーザー定義変数と同じです。 同じプリンタ番号に複数の論理名を割り当てることができます。 OUTPUT オペランド（下記参照）の値とは異なり、*logical-printer-name* はコンパイル時に評価されるので、プログラム制御フローから独立しています。
OUTPUT *operand1*	プリンタ名： *operand1* が使用できる場合は、そのフォーマット／長さを A8 または以下のいずれかにする必要があります。 名前は LPT*nn* として指定する必要があり、*nn* は 1～31 になります。 「*例 1*」も参照してください。 追加レポートには、次の名前を割り当てることができます。 **レポート** / **機能** DUMMY / 削除する出力。 INFOLINE / Natural 情報行への出力。 情報行の詳細については、Natural 端末コマンド %X（*端末コマンド*」ドキュメント）を参照してください。「*例 2*」も参照してください。 SOURCE / Natural ソースエリアへの出力。
PROFILE *operand2*	プリンタ制御文字テーブルの名前： PROFILE 節では、*operand2* としてプリンタ制御文字テーブルの名前を指定できます。 *operand2* の最大長は 8 です。 このようなテーブルは、グローバルコンフィグレーションファイルで定義します。 プリンタプロファイルを設定する方法の詳細については、「*プリンタプロファイル*」（『*コンフィグレーションユーティリティ*』ドキュメント）を参照してください。

例

例 1 - プリンタ名の定義

```
/* PRINTER NAME DEFINED FOR WINDOWS
*
DEFINE PRINTER (REPORT1 = 1) OUTPUT 'LPT1'
WRITE (REPORT1) 'REPORT 1 PRINTED ON PRINTER LPT1'
END
```

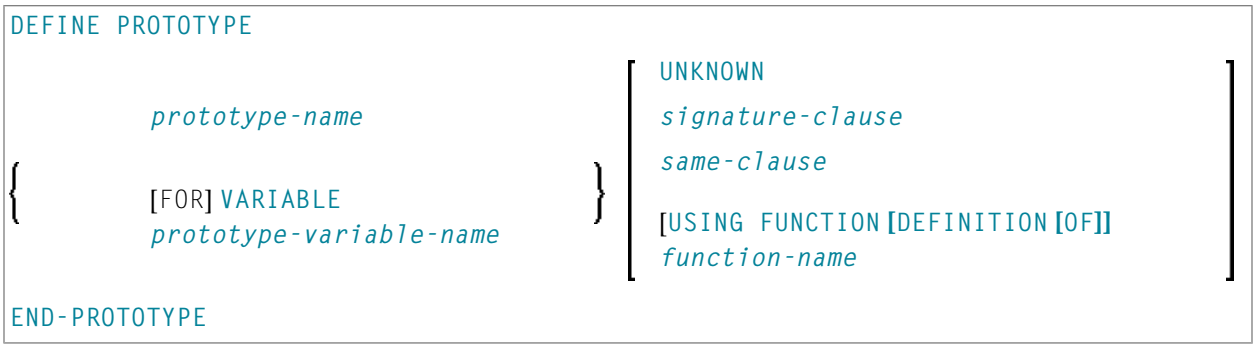
例 2 - 情報行への出力

```
** Example 'DPIEX1': DEFINE PRINTER
************************************************************************
*
SET CONTROL 'XI+'        /* SWITCH INFOLINE MODE ON
SET CONTROL 'XT'         /* INFOLINE TOP
*
DEFINE PRINTER (1) OUTPUT 'INFOLINE'
WRITE (1) 'EXECUTING' *PROGRAM 'BY' *INIT-USER
WRITE 'TEST OUTPUT'
EJECT                    /* FORCE PHYSICAL I/O
*
SET CONTROL 'X'          /* SWITCH BACK TO NORMAL
*
END
```

プログラム **DPIEX1** の出力：

```
EXECUTING DPIEX1   BY HTR
Page      1                                          05-01-13  14:54:33

TEST OUTPUT
```

49 DEFINE PROTOTYPE



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：DEFINE FUNCTION

『*プログラミングガイド*』の「*ユーザー定義関数*」も参照してください。

機能

プロトタイプ定義は、特定のファンクションコールに従って署名を指定するために使用することができます。 各ファンクションコールで、戻りタイプは、ファンクションコール（VARIABLE）の種類と同様に識別される必要があります。 したがって、このデータは各ファンクションコールに対して有効である必要があります。このデータがない場合、プロトタイプキーワードをファンクションコール参照内で使用する必要があります。 プロトタイプ内にパラメータ定義があれば、ファンクションコールのパラメータ値はプロトタイプ定義のパラメータと比較されます。 パラメータをチェックする必要がない場合は、プロトタイプ定義の DEFINE DATA PARAMETER ステートメント内で UNKNOWN キーワードを使用します。

関連トピック：

ユーザー定義関数
ファンクションコール

構文説明

prototype-name	*prototype-name*はユーザー定義変数を定義するために使用されるものと同じ規則に従う必要があります（例外：プロトタイプ名ではピリオド"."が使用可能）。 *prototype-name*は完全に任意です。 対応するファンクション定義と同じ名前にする必要はありません。 *prototype-name*全体の最大長は32文字です。
VARIABLE *prototype-variable-name*	*prototype-variable-name*により、可変のファンクション名を使用してファンクションを呼び出すことができます。 これはCALLNATファンクションの呼び出しに似ています。 *prototype-variable-name*は、ファンクション参照で呼び出されるファンクションの実際の名前を含む英数字変数名です。
UNKNOWN	パラメータをチェックする必要がない場合は、プロトタイプ定義のDEFINE DATA PARAMETERステートメント内でUNKNOWNキーワードを使用します。
signature-clause	下記の「***signature*** *節*」を参照してください。
prototype-return-data-definition	「*プロトタイプ戻りデータ定義*」を参照してください。
same-clause	下記の「***SAME AS*** *節*」を参照してください。
USING FUNCTION [DEFINITION [OF]] *function-name*	下記の「***USING FUNCTION*** *節*」を参照してください。
END-PROTOTYPE	DEFINE PROTOTYPEステートメントを終了するには、Natural予約語END-PROTOTYPEを使用する必要があります。

signature 節

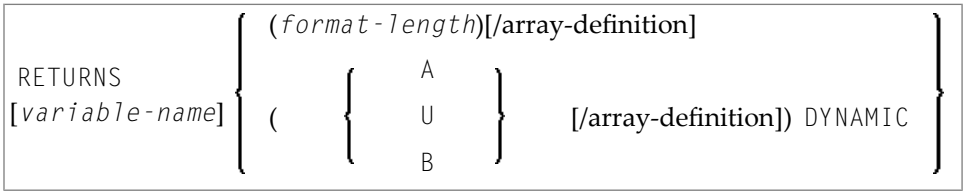
```
    [prototype-return-data-definition]
[   DEFINE DATA
 { PARAMETER UNKNOWN                                              }
 {                                                                }
 { PARAMETER                  { USING parameter-data-area       } ... }
 {                            { parameter-data-definition       }      }
    END-DEFINE                                                    ]
```

signature（署名）節はある種のファンクションコールのように見えます。 通常、プロトタイプはファンクション定義に一致します。 しかし、正確に同じである必要はありません。 したがって、パラメータデータを省略したり、代わりにキーワードUNKNOWNを設定することが可能です。 この場合、コンパイル時にチェックされるパラメータはありません。

戻り値のタイプは、どのような場合でも設定する必要があります。 戻り値を定義していない場合は、ファンクションコールから変数への割り当てを行うことはできません。

プロトタイプ定義内で署名を指定していない場合（署名が UNKNOWN の場合）は、ファンクションコールの対応する署名をキーワード PT を使用して指定する必要があります。 PT の詳細については、『*プログラミングガイド*』の「*ファンクションコール*」にある「*プロトタイプキャスト*」セクションを参照してください

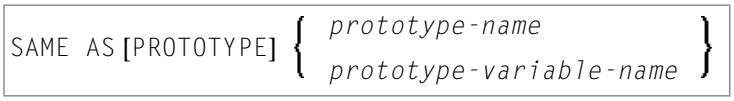
プロトタイプ戻りデータ定義



この節では、コンパイル時に認識される必要がある戻り値の *format-length* を定義します。

オプションの変数名は無視されます。 このオプションは、DEFINE FUNCTION ステートメントの RETURNS 節と同じ構文構造にするために導入されています。

SAME AS 節



この節では、以前に定義した署名を使用して新規のプロトタイプを定義します。

USING FUNCTION 節

この明示的な節を使用すると、ファンクションのパラメータ定義に対して生成されたオブジェクトの分析が可能になります。この分析は、そのファンクションの論理名で間接的な DEFINE PROTOTYPE ステートメントを作成するために実行されます。 *function-name* は論理名ですが、ファンクションオブジェクトのオブジェクト名ではありません。 論理ファンクション名は、対応するファンクションオブジェクトのファンクションの本体 DEFINE FUNCTION *function-name* ... END-FUNCTION で定義します。

例

例 1 - DEFINE PROTOTYPE

これは "GET-FIRST-BYTE" というファンクションのプロトタイプ定義です。 次のプロトタイプを使用すると、ファンクション "GET-FIRST-BYTE" を記号ファンクションコールとして呼び出すことができます。

```
GET-FIRST-BYTE(<#A>)
```

```
DEFINE DATA LOCAL
1 #A(A10) INIT <'abcdefghij'>
END-DEFINE

DEFINE DATA LOCAL
1 #A(A10) INIT <'abcdefghij'>
END-DEFINE

DEFINE PROTOTYPE GET-FIRST-BYTE
  RETURNS (A1)
  DEFINE DATA PARAMETER
  1 PARM1(A10)
  END-DEFINE
END-PROTOTYPE

WRITE GET-FIRST-BYTE(<#A>)
END
```

例 2 - DEFINE PROTOTYPE

次のNaturalコードには、ファンクションのプロトタイプ定義"GET-FIRST-BYTE"が含まれています。 ファンクションをダイナミックに呼び出すことができるように、ファンクションの名前は英数字変数#Aに保存する必要があります。 変数#Aは、使用する前に英数字変数として DEFINE DATA ステートメント内で定義する必要があります。

```
DEFINE DATA LOCAL
1 FUNCTION-NAME(A32) INIT<'GET-FIRST-BYTE'>
1 #A(A10) INIT <'abcdefghij'>
END-DEFINE

DEFINE PROTOTYPE VARIABLE FUNCTION-NAME
  RETURNS (A1)
  DEFINE DATA PARAMETER
  1 PARM1(A10)
  END-DEFINE
END-PROTOTYPE

WRITE FUNCTION-NAME(<#A>)
END
```

50 DEFINE SUBROUTINE

```
DEFINE [SUBROUTINE] subroutine-name
              statement ...
{             END-SUBROUTINE                    }
{             RETURN（レポーティングモードのみ）  }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CALL** | **CALL FILE** | **CALL LOOP** | **CALLNAT** | **ESCAPE** | **FETCH** | **PERFORM**

関連機能グループ：「*プログラムおよびルーチンの呼び出し*」

機能

DEFINE SUBROUTINE ステートメントでは、Naturalサブルーチンを定義します。サブルーチンは PERFORM ステートメントによって呼び出されます。

内部／外部サブルーチン

サブルーチンは、PERFORM ステートメントを含むオブジェクトの中でも（内部サブルーチン）、PERFORM を含むオブジェクトの外でも（外部サブルーチン）定義できます。 内部サブルーチンは、それを参照する最初の PERFORM ステートメントの前/後いずれにでも定義できます。

Note: プログラム機能を複数の外部サブルーチンで構築すると、プログラム構造が明確になります。ただし、外部サブルーチンの呼び出しは、内部コードまたは内部サブルーチンに比べてオーバーヘッドが多くなるため、常に外部サブルーチンにはできるだけ大きな機能ブロックを含めるようにすることをお勧めします。

制限事項

- サブルーチン内で開始した処理ループは、すべて END-SUBROUTINE を発行する前に閉じる必要があります。
- 内部サブルーチンの中で、別の DEFINE SUBROUTINE ステートメントを指定することはできません（下記の「*例 1*」を参照）。
- 外部サブルーチン（サブルーチンタイプのオブジェクト）に、複数の DEFINE SUBROUTINE ステートメントブロックを指定することはできません（下記の「*例2*」を参照）。ただし、外部の DEFINE SUBROUTINE ブロックの中に内部サブルーチンを指定することはできます（下記の「*例 1*」を参照）。

例 **1**

次のような構造は、タイプがサブルーチンのオブジェクトでは有効ですが、その他のオブジェクト（SUBR01 が内部サブルーチンとみなされます）の場合は無効です。

```
...
DEFINE SUBROUTINE SUBR01
   ...
   PERFORM SUBR02
   PERFORM SUBR03
   ...
   DEFINE SUBROUTINE SUBR02
   /* inline subroutine...
   END-SUBROUTINE
 ...
   DEFINE SUBROUTINE SUBR03
   /* inline subroutine...
   END-SUBROUTINE
END-SUBROUTINE
END
```

例 **2**（無効）

次の構造は、オブジェクトタイプがサブルーチンの場合には*無効*です。

```
...
DEFINE SUBROUTINE SUBR01
...
END-SUBROUTINE
DEFINE SUBROUTINE SUBR02
...
END-SUBROUTINE
END
```

構文説明

subroutine-name	サブルーチン名（最大 32 文字）の命名規則は、ユーザー定義変数名の規則と同じです。『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。 サブルーチン名は、該当のサブルーチンが定義されているモジュールの名前に依存しません（同じ名前にすることは可能ですが、同じ名前にする必要はありません）。
END-SUBROUTINE	サブルーチン定義は END-SUBROUTINE で終了します。
RETURN	レポーティングモードでは、RETURN でもサブルーチンを終了できます。

サブルーチンで使用可能なデータ

このsectionでは、次のトピックについて説明します。

インラインサブルーチン

明示的なパラメータを、呼び出し元プログラムから PERFORM ステートメントを介して内部サブルーチンに渡すことはできません。

内部サブルーチンからは、呼び出し元のプログラム内で定義してあるローカルデータエリアおよびグローバルデータエリアのデータにアクセスできます。

外部サブルーチン

外部サブルーチンは、現在設定されているグローバルデータエリアにアクセスできます。また、PERFORMステートメントで、呼び出し元のオブジェクトから外部サブルーチンにパラメータを直接渡せるため、グローバルデータエリアのサイズを縮小できます。

外部サブルーチンでは、呼び出し元のプログラムで定義したローカルデータエリアは使用できませんが、独自のローカルデータエリアを持つことはできます。

例

例 1 - サブルーチンの定義

```
** Example 'DSREX1S': DEFINE SUBROUTINE (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE  (A20/2)
  2 PHONE
*
1 #ARRAY    (A75/1:4)
1 REDEFINE #ARRAY
  2 #ALINE  (A25/1:4,1:3)
```

```
1 #X        (N2) INIT <1>
1 #Y        (N2) INIT <1>
END-DEFINE
*
FORMAT PS=20
LIMIT 5
FIND EMPLOY-VIEW WITH NAME = 'SMITH'
  MOVE NAME            TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE           TO #ALINE (#X+3,#Y)
  IF  #Y = 3
    RESET INITIAL #Y
    PERFORM PRINT
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
    PERFORM PRINT
  END-ENDDATA
END-FIND
*
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
END-SUBROUTINE
*
END
```

プログラム **DSREX1S** の出力：

```

SMITH                    SMITH                    SMITH
ENGLANDSVEJ 222          3152 SHETLAND ROAD       14100 ESWORTHY RD.
                         MILWAUKEE                MONTERREY
554349                   877-4563                 994-2260

SMITH                    SMITH
5 HAWTHORN               13002 NEW ARDEN COUR
OAK BROOK                SILVER SPRING
150-9351                 639-8963
```

レポーティングモードの例については、ライブラリ SYSEXRM のプログラム **DSREX1R** を参照してください。

例 2 - GDA フィールドを使用する外部サブルーチンのサンプル構造

```
** Example 'DSREX2': DEFINE SUBROUTINE (using GDA fields)
************************************************************************
DEFINE DATA
GLOBAL
  USING DSREX2G
END-DEFINE
*
INPUT 'Enter value in GDA field' GDA-FIELD1
*
*  Call external subroutine in DSREX2S
*
PERFORM DSREX2-SUB
*
END
```

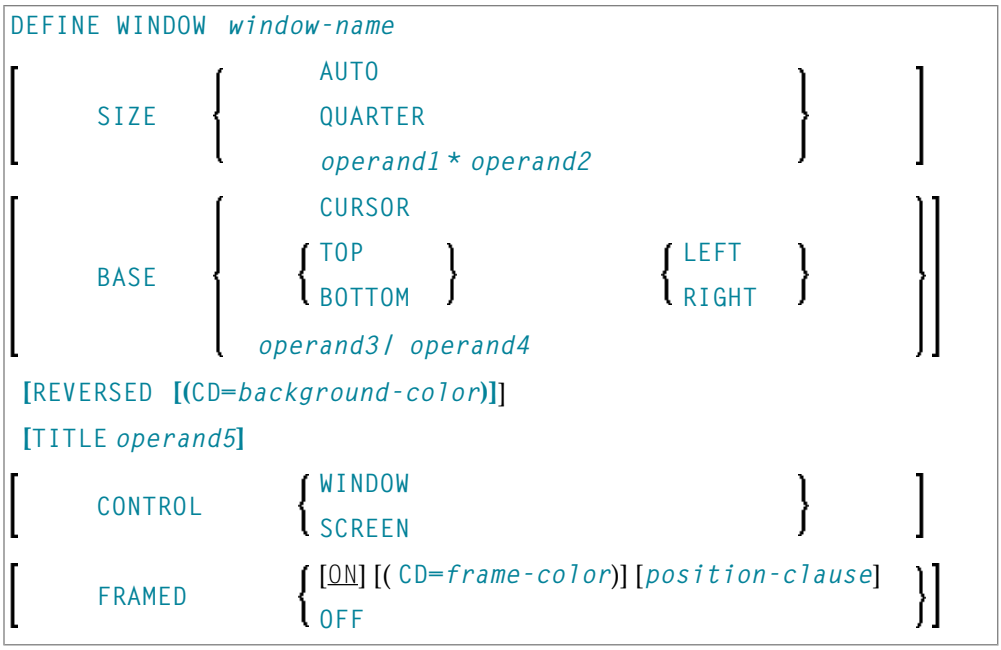
プログラム **DSREX2** によって使用されるグローバルデータエリア **DSREX2G**：

```
1 GDA-FIELD1                        A    2
```

プログラム **DSREX2** によって呼び出されるサブルーチン **DSREX2S**：

```
** Example 'DSREX2S': SUBROUTINE (external subroutine using global data)
************************************************************************
DEFINE DATA
GLOBAL
  USING DSREX2G
END-DEFINE
*
DEFINE SUBROUTINE DSREX2-SUB
*
  WRITE 'IN SUBROUTINE' *PROGRAM '=' GDA-FIELD1
*
END-SUBROUTINE
*
END
```

51 DEFINE WINDOW



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DEFINE WINDOW** | **INPUT** | **REINPUT** | **SET WINDOW**

関連機能グループ：「*対話型処理用の画面生成*」

機能

DEFINE WINDOW ステートメントでは、ウィンドウのサイズ、位置、属性を指定します。

ウィンドウとは、端末画面上に表示される、プログラムによって構築された論理ページのセグメントのことです。 存在を認識できなくても、常にウィンドウは存在してます。別のウィンドウが指定されない限り、ウィンドウのサイズは端末画面の物理サイズと同じです。

DEFINE WINDOW ステートメントはウィンドウをアクティブ化しません。ウィンドウをアクティブ化するには、SET WINDOW ステートメントを使用するか、INPUT ステートメントの WINDOW 節を使用します。

Note: Natural ウィンドウ、つまり最新のウィンドウは常に *1* つです。 画面に前のウィンドウが表示されていても、アクティブではないので、Natural に無視されます。 現在のウィンドウにのみ入力できます。 入力する十分なスペースがない場合は、まずウィンドウサイズを調整する必要があります。

フルスクリーンの制御

ウィンドウがアクティブな場合でも、Naturalによってフルスクリーンの制御が行われます。これはCICSやTSOのようなシングルセッションシステムに影響を与えません。Naturalが Completeや Multi-passのようなマルチプルセッションシステムの環境下で動作している場合、中断されたNaturalセッションが再開すると、完全なNatural画面（つまり現在アクティブなウィンドウだけでなく、「その下」 のNatural画面も含む）が表示されます。同時に、フルスクリーンのフィールド属性はウィンドウによって部分的にオーバーレイされますが、ウィンドウによる影響はありません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											オペランド参照	ダイナミック定義
operand1	C	S						N	P	I							可	不可
operand2	C	S						N	P	I							可	不可
operand3	C	S						N	P	I							可	不可
operand4	C	S						N	P	I							可	不可
operand5	C	S				A	U										可	不可

構文要素の説明：

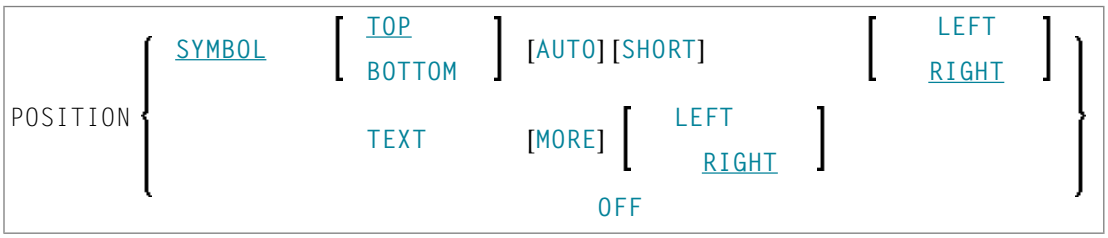
window-name	*window-name* では、ウィンドウの名前を指定します。 名前の最大長は 32 文字です。 ウィンドウ名には、ユーザー定義変数と同じ命名規則が適用されます。『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。
SIZE	SIZE 節では、ウィンドウのサイズを指定します。 **注意:** メインフレーム環境では、Natural でデータを画面上に表示できるようにするには属性バイトと呼ばれる追加の列が必要です（他のプラットフォームでは、そのような属性バイトは不要）。したがって、メインフレーム環境では他のプラットフォームよりもウィンドウによってオーバーレイされる画面領域が大きく、ウィンドウ内に表示できるページセグメントのサイズが小さくなります。 例：ウィンドウサイズを SIZE 5 * 15（15 桁の幅）として定義します。 ■ メインフレーム環境では、ウィンドウによってオーバーレイされる画面領域は16桁で、ウィンドウ内に表示できるサイズはフレームなしで14桁、フレーム付きで 10 桁です。 ■ 他のプラットフォームでは、ウィンドウによってオーバーレイされる画面領域は 15 桁で、ウィンドウ内に表示できるサイズはフレームなしで 15 桁、フレーム付きで 13 桁です。

SIZE AUTO	ウィンドウのサイズは、Natural 実行時に自動で決定されます。 サイズは、ウィンドウに生成されるデータによって、次のように決定されます。 ■ ウィンドウの行数は、生成される INPUT 行の数です（PF キー行、メッセージ行、および情報／統計行を加えることが可能）。 ■ ウィンドウの桁数は、最も長い INPUT 行で決まります。Natural では、行の終わりから行の有効バイトの右端までが検索されます。 これにより、入力だけのフィールド（AD=A）または修正可能フィールド（AD=M）は、切り捨てられる可能性があります。これを回避するには、このようなフィールドの後に単一文字を置くか、または次の節でウィンドウサイズを明示的に設定します。 SIZE *operand1* * *operand2* SIZE 節を省略すると、デフォルトで SIZE AUTO が適用されます。 **注意:** タイトルはウィンドウデータの一部ではありません。 したがって、ウィンドウサイズが上記のように指定され、さらにタイトルがウィンドウよりも長い場合、タイトルは切り捨てられます。
SIZE QUARTER	ウィンドウサイズは物理画面の 1/4 になります。
SIZE *operand1* * *operand2*	ウィンドウサイズは *n* 行 × *n* 桁になります。 *operand1* では行数、*operand2* では桁数を指定します。 2 つのオペランドのいずれにも小数桁を含めることはできません。 FRAMED を指定している場合、指定サイズにはフレームも含まれます。 有効な最小ウィンドウサイズは、次のようになります。 ■ フレームなしの場合は 2 行 × 10 桁 ■ フレーム付きの場合は 4 行 × 13 桁 有効な最大ウィンドウサイズは、物理画面のサイズです。
BASE	BASE 節では、物理画面上でのウィンドウ位置を決定します。 BASE 節を省略すると、デフォルトで BASE CURSOR が適用されます。
BASE CURSOR	ウィンドウの左上隅を、現在のカーソル位置に配置します。 カーソル位置とは、画面上の物理的なカーソルの位置です。 ウィンドウサイズが原因でウィンドウをカーソル位置に配置できない場合、Natural では、目的の位置にできるだけ近くなるようにウィンドウが自動的に配置されます。
BASE TOP/BOTTOM LEFT/RIGHT	それぞれ、物理画面上の左上隅、左下隅、右上隅、右下隅にウィンドウを配置します。
BASE *operand3*/*operand4*	物理画面の指定した行／桁に、ウィンドウの左上隅を配置します。 *operand3* では行数、*operand4* では桁数を指定します。 2 つのオペランドのいずれにも小数桁を含めることはできません。 ウィンドウサイズが原因で指定の位置にウィンドウを配置できない場合は、エラーメッセージが返されます。
REVERSED	REVERSED を指定すると、ウィンドウが反転表示されます。ただし、この指定は、使用画面でこの機能がサポートされている場合にのみ有効となります。サポートされていない場合、REVERSED は無視されます。

REVERSED CD= *background-color*	ウィンドウを反転表示し、さらにウィンドウの背景を指定の色で表示します。ただし、この指定は、使用画面でこれらの機能がサポートされている場合にのみ有効となります。サポートされていない場合、各指定は無視されます。 有効なカラーコードについては、『パラメータリファレンス』でセッションパラメータ CD を参照してください。
TITLE *operand5*	TITLE 節では、ウィンドウの見出しを指定できます。 指定したタイトル（*operand5*）は、ウィンドウの先頭フレーム行の中央に表示されます。 タイトルは、テキスト文字列（アポストロフィで囲む）、またはユーザー定義変数の内容として指定できます。 タイトルがウィンドウよりも長い場合、タイトルは切り捨てられます。タイトルは、FRAMED を指定した場合にのみ表示されます。FRAMED OFF を指定すると、TITLE 節は無視されます。 **注意:** タイトルの末尾にある空白は削除されます。 タイトルの先頭にある文字が空白の場合は、自動的に 1 桁の空白がタイトルに付加されます。
CONTROL	CONTROL 節を使用して、PF キー行、メッセージ行、および統計行をウィンドウに表示するかフル物理画面に表示するかを指定します。
CONTROL WINDOW	CONTROL WINDOW を指定すると、ウィンドウ内に PF キー行、メッセージ行、および統計行が表示されます。 CONTROL 節を省略すると、デフォルトで CONTROL WINDOW が適用されます。
CONTROL SCREEN	CONTROL SCREEN を指定すると、ウィンドウの外側にある物理画面全体に PF キー行、メッセージ行、および統計行が表示されます。
FRAMED	デフォルトでは、つまり FRAMED 節を省略した場合は、ウィンドウにフレームが付きます。 最上部と最下部のフレーム行はカーソルに依存します。適用可能なところで、適切な記号（<、-、+、>：下記の *position-clause* を参照）の上にカーソルを置いて Enter キーを押すと、ウィンドウ内でページを前後左右に移動できます。記号が表示されない場合は、カーソルを枠線の最上部（前のページに戻る場合）または最下部（次のページに進む場合）に置いて Enter キーを押すことにより、ウィンドウ内でページを前後に移動することができます。 **注意:** ウィンドウサイズが 4 行 × 12（メインフレーム環境では 13）桁より小さい場合、フレームは表示されません。
FRAMED OFF	FRAMED OFF を指定すると、フレーム表示および各フレーム関連の指定（ウィンドウタイトルおよび位置情報）はオフになります。
FRAMED (CD= *frame-color*)	この節では、ウィンドウに表示するフレームの色を指定します。ただし、使用画面がカラー端末でない場合、カラー指定は無視されます。 有効なカラーコードについては、『パラメータリファレンス』でセッションパラメータ CD を参照してください。 **注意:** Natural for Windows では、この指定は無視されます。
position-clause	POSITION 節は、メインフレームのコンピュータだけで評価されます。他のすべてのプラットフォームでは無視されます。 詳細については、下記の「***POSITION 節***」を参照してください。

POSITION 節

POSITION 節は、メインフレームのコンピュータだけで評価されます。他のすべてのプラットフォームでは無視されます。



POSITION 節を指定すると、論理ページ上でのウィンドウの位置に関する情報がウィンドウのフレーム内に表示されます。 これは、論理ページがウィンドウよりも大きい場合にのみ適用されます。そうでない場合は、POSITION 節は無視されます。 位置情報は、論理ページが拡張する方向（現在のウィンドウの上、下、左、右）を示します。

POSITION 節を省略すると、デフォルトで POSITION SYMBOL TOP RIGHT が適用されます。

POSITION SYMBOL	位置情報を記号形式 "More: <-+>" で表示します。 情報は、上部か下部のいずれか、または両方のフレームラインに表示されます。
TOP/BOTTOM	位置情報を上のフレーム行または下のフレーム行のどちらに表示するかを決定します。
AUTO	論理ページの横のサイズがウィンドウ内に納まる場合に限り適用されます。つまり、"-" または "+" が表示される場合にのみ適用されます。 この場合、AUTO によって、記号はそれぞれ文字列 "Top"、"Bottom"、"More" に自動で切り替えられます。
SHORT	記号 "<-+>" の前にある文字列 "More:" を非表示にします。
LEFT/RIGHT	位置情報をフレーム行の左または右のどちらに表示するかを決定します。
POSITION TEXT	位置情報をテキスト形式で表示します。 情報を上か下のフレーム行に "More"、"Top"、"Bottom" の文字列で表示します。 このテキストは、言語依存であり、言語コードによって別の言語でも表示できます。
POSITION TEXT MORE	文字列 "Top" と "Bottom" を非表示にして、適用される場所（上、下またはその両方のフレーム行）に文字列 "More" のみを表示します。
LEFT/RIGHT	位置情報をフレーム行の左または右のどちらに表示するかを決定します。
POSITION OFF	位置情報を非表示にします。位置情報は表示されません。

ウィンドウの入力フィールドの保護

全体をウィンドウ内に収容できない入力フィールド（AD=A または AD=M）には、次の規則が適用されます。

- フィールドの始まりがウィンドウ内にない入力フィールドは常に保護されます。
- ウィンドウ内で始まりウィンドウ外で終わる入力フィールドは、フィールドに含まれる値がウィンドウ内に完全に表示できない場合にのみ保護されます。 この場合、*フィールド長*ではなく*値の長さ*がウィンドウのサイズを超えているかどうかを決定します。充填文字（プロファイルパラメータ FC で指定）は、値の一部には数えられません。

保護された入力フィールドにアクセスする場合は、フィールドの先頭および値の終わりがウィンドウ内に入るようにウィンドウサイズを調整する必要があります。

他のウィンドウの呼び出し

DEFINE WINDOW ステートメントは、論理条件ステートメントブロック内では指定できません。条件によって異なるウィンドウを呼び出すには、条件内で別の SET WINDOW ステートメント（または WINDOW 節を含む INPUT ステートメント）を使用します。

例

```
** Example 'DWDEX1': DEFINE WINDOW
************************************************************************
DEFINE DATA LOCAL
01 #I (P3)
END-DEFINE
*
SET KEY PF1='%W<<' PF2='%W>>' PF4='%W--' PF5='%W++'
*
DEFINE WINDOW WIND1
       SIZE QUARTER
       BASE TOP RIGHT
       FRAMED ON POSITION SYMBOL AUTO
*
SET WINDOW 'WIND1'
FOR #I = 1 TO 10
  WRITE 25X #I 'THIS IS SOME LONG TEXT' #I
END-FOR
```

```
*
END
```

プログラム **DWDEX1** の出力：

```
                                      +------------------------More:     + >+
> r                                   ! Page      1                         !
All    ....+....1....+....2....+....3.. !                                   !
  0010 ** Example 'DWDEX1': DEFINE WIND !                             1 THIS  !
  0020 ******************************** !                             2 THIS  !
  0030 DEFINE DATA LOCAL                !                             3 THIS  !
  0040 01 #I (P3)                       !                             4 THIS  !
  0050 END-DEFINE                       !                             5 THIS  !
  0060 *                                !                             6 THIS  !
  0070 SET KEY PF1='%W<<' PF2='%W>>' PF !                             7 THIS  !
  0080 *                                ! MORE                              !
  0090 DEFINE WINDOW WIND1              +-------------------------------------+
  0100        SIZE QUARTER
  0110        BASE TOP RIGHT
  0120        FRAMED ON POSITION SYMBOL AUTO
  0130 *
  0140 SET WINDOW 'WIND1'
  0150 FOR #I = 1 TO 10
  0160   WRITE 25X #I 'THIS IS SOME LONG TEXT' #I
  0170 END-FOR
  0180 *
  0190 END
  0200
       ....+....1....+....2....+....3....+....4....+....5....+... S 19   L 1
```

52 DEFINE WORK FILE

```
DEFINE WORK FILE n [operand1] [TYPE operand2] [ATTRIBUTES {operand3}...]
```

Note: 角カッコ [...] で示されている要素はオプションですが、このステートメントではこれらのオプションの少なくとも 1 つを指定する必要があります。

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CLOSE WORK FILE** | **READ WORK FILE** | **WRITE WORK FILE**

関連機能グループ：「*ワークファイル／PC ファイルの制御*」

機能

DEFINE WORK FILE ステートメントは、Natural アプリケーション内で Natural ワークファイル番号にファイル名を割り当てるために使用します。

このステートメントを使用することで、Naturalセッション内でワークファイルの割り当てをダイナミックに行ったり、変更することができます。また、別のレベルで作成したワークファイルの割り当てを上書きすることもできます。

このステートメントを実行したときに、指定したワークファイルがすでに開かれていると、ワークファイルはこのステートメントによって暗黙的に閉じられます。

Note: Unicode およびコードページのサポートについては、『*Unicode およびコードページのサポート*』ドキュメントの「*Windows、UNIX、および OpenVMS プラットフォーム上のワークファイルと出力ファイル*」を参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット										オペランド参照	ダイナミック定義
operand1	C	S				A	U									可	不可
operand2	C	S				A	U									可	不可
operand3	C	S				A	U									可	不可

Note: フォーマット U オペランドを Unicode（UTF-16）で指定すると、このオペランドは評価の前にセッションコードページ文字に変換されます。

構文要素の説明：

DEFINE WORK FILE *n*	ワークファイル番号： *n* は、ワークファイル番号です（1～32）。 これは、WRITE WORK FILE、READ WORK FILE、または CLOSE WORK FILE ステートメントのいずれかで使用される番号です。	
operand1	ワークファイル名： *operand1* は、ワークファイルの名前です。 ファイル名（*operand1*）には、環境変数を含めることもできます。 物理ワークファイル名を使用することも可能です。指定した名前のファイルが存在しない場合は、新規のファイルが作成されます。 *operand1* を指定しない場合は、コンフィグレーションユーティリティで対応するワークファイル番号のパラメータファイルに保存されたワークファイル名によって、*operand1* の値が決定されます。 **注意:** *operand1* が指定されていない場合、Natural for Mainframes と Natural for Windows/UNIX/OpenVMS の動作は異なります。	
TYPE *operand2*	**TYPE** 節： *operand2* はワークファイルのタイプを指定します。 *operand2* の値では、大文字と小文字が区別されません。値は引用符で囲むか、英数字の変数で指定する必要があります。	
	DEFAULT	拡張子からファイルタイプを特定します。 フォーマット：ワークファイルタイプによって異なります。 **注意:** ファイルタイプ TRANSFER は、ワークファイルタイプ DEFAULT によって特定することができません。 使用するファイルタイプとして、TRANSFER を明示的に定義する必要があります。
	TRANSFER	Entire Connection と PC の間で両方向のデータ転送を行うために使用します。 このワークファイルタイプは、UNIX または OpenVMS 上の Natural セッションと PC 上の Entire Connection 末端の間のデータ接続を示します。 フォーマット：ENTIRE CONNECTION **注意:** 1. このワークファイルは、***ATTRIBUTES 節***と一緒に使用することはできません。 2. このワークファイルタイプは、Windows 環境では利用できません。

	SAG	フォーマット：バイナリ
	ASCII	ASCII フォーマットのファイルは、[キャリッジリターン] 改行で終了するレコードを含む 「テキスト」 ファイルです。 フォーマット：ASCII
	ASCII-COMPRESSED	ASCII フォーマットのファイルですが、通常の ASCII フォーマットとは異なり、末尾にある空白はすべて削除されます。 フォーマット：ASCII
	ENTIRECONNECTION	このワークファイルタイプでは、ローカルディスクにある Entire Connection フォーマットのワークファイルに対して直接読み取りおよび書き込みを行うことができます（例えば、ステートメント `READ` と `WRITE` を使用）。 フォーマット：ENTIRE CONNECTION **注意:** このワークファイルタイプは、PC、UNIX、および OpenVMS 上で利用できます。 PC へは転送できません。 Entire Connection 末端はこの処理では使用されません。
	UNFORMATTED	フォーマットがまったく行われていないファイル。フォーマット情報が書き込まれていません（フィールドまたはレコードのいずれにも）。 フォーマット：UNFORMATTED
	PORTABLE	正確にダイナミック変数を操作できるファイルで転送することも可能です（例：リトルエンディアンマシンとビッグエンディアンマシンの間）。 フォーマット：PORTABLE
	CSV	コンマ区切りの値。各レコードは、ファイル内の 1 つの行に書き込まれます。デフォルトでは、ヘッダーは書き込まれません。データフィールドを区切るために使用されるデフォルト文字はセミコロン（";"）です。 詳細については、『*コンフィグレーションユーティリティ*』の「*ワークファイル設定*」を参照してください。
ATTRIBUTES **{***operand3***}...**	**ATTRIBUTES** 節： *operand3* では、ワークファイル属性を指定します。	

複数の属性を、次の例のようにコンマまたは空白で区切って指定できます。

```
DEFINE WORK FILE ATTRIBUTES 'APPEND,KEEP'
```

同じ属性タイプの値を複数指定した場合は、次の例のように最後の値が使用されます。

```
DEFINE WORK FILE ATTRIBUTES 'APPEND,NOAPPEND'
```

この例では、NOAPPEND が使用されます。

BOM/NOBOM の使用例：

```
...
DEFINE WORK FILE 11 'x.tmp' ATTRIBUTES 'BOM'
*
* write work file with BOM
*
DEFINE WORK FILE 11 'x.tmp' ATTRIBUTES 'NOBOM'
*
* write work file without BOM
...
```

注意: *operand3*を省略した場合は、パラメータファイル（コンフィグレーションユーティリティで作成された）で定義されている対応する値が暗黙的に使用されます。

次の表に、属性タイプの概要および設定可能な値を示します。

追加モードの有効／無効：	
APPEND	追加モードを有効にします。このモードでは、新規レコードはファイルの末尾に追加されます。
NOAPPEND	追加モードを無効にします。ファイルは先頭から再書き込みされます。 これはデフォルト値です。
書き込み BOM モードの有効／無効：	
BOM	書き込み BOM（バイトオーダマーク）モードを有効にします。 ワークファイルデータの直前に BOM が書き込まれます。
NOBOM	書き込み BOM モードを無効にします。 BOM は書き込まれません。 これはデフォルト値です。
ワークファイルを閉じた後のファイルの保持／削除：	
DELETE	ワークファイルを閉じる操作が完了すると、ワークファイルは削除されます。
KEEP	ワークファイルを閉じる操作が完了した後も、ワークファイルは保持されます。 これはデフォルト値です。

ワークファイルの詳細については、「*ワークファイルフォーマット*」を参照してください。

53 DELETE

```
DELETE [RECORD] [IN] [STATEMENT] [(r)]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

DELETE ステートメントは、データベースからレコードを削除するために使用します。

ホールド状態

DELETE ステートメントを使用すると、対応する FIND や READ ステートメントで選択した各レコードはホールド状態に置かれます。

レコードのホールドロジックの詳細については、『*プログラミングガイド*』の「*データベース更新 - トランザクション処理*」セクションを参照してください。

制限事項

DELETE ステートメントは、FIND、READ、または GET ステートメントと同じ行に指定することはできません。

構文説明

(*r*)	ステートメント参照： 表記 (*r*) を指定して、削除するレコードの選択と読み取りに使用されたステートメントを参照します。 (r) を指定しなかった場合、DELETE ステートメントは、データベースレコードが選択されて読み込まれた最も内側でアクティブになっている処理ループを参照します。

データベース固有の考慮事項

SQL データベース	DELETE ステートメントは、データベーステーブルから 1 つの行を削除するために使用します。 これは、SQL ステートメントの DELETE WHERE CURRENT OF CURSOR-NAME に対応するものです。つまり、最後に読み取られた行のみを削除できます。 ほとんどの SQL データベースでは、FIND SORTED BY または READ LOGICAL ステートメントで読み取られた行は削除できません。
XML データベース	DELETE ステートメントは、データベースから XML オブジェクトを削除するために使用します。 XML データベースでは、最後に読み取られたレコードのみを削除できます。

例

例 1

この例では、ALDEN を含むレコードがすべて削除されます。

```
** Example 'DELEX1': DELETE
**
**
CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
  /*
  DELETE
  END TRANSACTION
  /*
  AT END OF DATA
    WRITE NOTITLE *NUMBER 'RECORDS DELETED'
  END-ENDDATA
```

```
END-FIND
END
```

例 2

VEHICLESファイルでALDENという名前の人物が見つからない場合は、ALDENのEMPLOYEEレコードが削除されます。

```
** Example 'DELEX2': DELETE
**
**
CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
END-DEFINE
*
EMPL. FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
  /*
 VEHC. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMPL.)
    IF NO RECORDS FOUND
      /*
      DELETE (EMPL.)
      /*
      END TRANSACTION
    END-NOREC
  END-FIND
  /*
END-FIND
END
```

54 DISPLAY

```
DISPLAY [( rep)] [options ] {[ / ... ] [output-format ] output-element} ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

DISPLAY ステートメントでは、レポートに出力するフィールドを列形式で指定します。 列は各フィールドごとに作成され、フィールドヘッダーが列の上に表示されます。

Note: WRITE ステートメントと PRINT ステートメントを使用すると、フリーフォーマット（列なし）の出力を生成することができます。

『*プログラミングガイド*』の次のトピックも参照してください。

- *データ出力制御*
- *DISPLAY および WRITE ステートメント*
- *マルチプルバリューフィールドとピリオディックグループのインデックス表記*
- *列ヘッダー*
- *出力ページのレイアウト*

構文説明

(*rep*)	レポート指定： 表記 (*rep*) を使用すると、DISPLAY ステートメントを適用するレポートの ID を指定できます。 レポート ID として、範囲 0～31 の値、または DEFINE PRINTER ステートメントを使用して割り当てた論理名を指定できます。 (*rep*) を指定しない場合、ステートメントは最初のレポート（レポート 0）に適用されます。

	プリンタファイルが PC として Natural に定義されている場合、レポートは PC にダウンロードされます。「*例 8*」を参照してください。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。
options	表示オプション： 詳細については、下記の「*表示オプション*」を参照してください。
output-format	出力フォーマット定義： 詳細については、下記の「*出力フォーマット定義*」を参照して下さい。
/	行送り - スラッシュ表記： 文字列内でスラッシュ（"/"）を指定すると、表示中の文字列が改行されます。 出力項目の間にスラッシュを指定すると、"/" で指定された出力項目は、同じ列内で縦方向に配置されます。 この列のヘッダーでは、縦方向に表示されている項目のヘッダーが、列の上側に縦方向で配置されます。 『*プログラミングガイド*』の次のトピックも参照してください。 ■ *行送り - スラッシュ表記* ■ *例 1 - DISPLAY ステートメントでの改行* ■ *列ヘッダーの省略 - スラッシュ表記*
output-element	出力項目： 詳細については、下記の「*出力項目*」を参照してください。

表示オプション

```
[NOTITLE] [NOHDR] [ [AND] [GIVE] [SYSTEM] FUNCTIONS ] [(statement-parameters)]
```

構文要素の説明：

NOTITLE	デフォルトページタイトルの省略： デフォルトでは、DISPLAY ステートメントで出力される各ページに、Natural によって 1 行のタイトル行が作成されます。 このタイトルには、ページ番号と日時が含まれます。 時刻は、プログラム実行開始時またはジョブ開始時（バッチモード）に設定されます。 このデフォルトタイトル行は、WRITE TITLE ステートメントで書き換えることができます。また、DISPLAY ステートメントにキーワード NOTITLE を指定して、タイトル行を省略することもできます。 例：

	■ デフォルトタイトルが生成されます。 `DISPLAY NAME` ■ ユーザータイトルが生成されます。 `DISPLAY NAME WRITE TITLE '`*`user-title`*`'` ■ タイトルは生成されません。 `DISPLAY NOTITLE NAME` **注意:** NOTITLE オプションを使用すると、同じオブジェクト内でデータを同じレポートに書き込むすべての DISPLAY、PRINT、および WRITE ステートメントに適用されます。
NOHDR	列ヘッダー： 列ヘッダーは、DISPLAY ステートメントで指定した各フィールド（列）ごとに、次の規則に従って生成されます。 ■ DISPLAY ステートメントでは、フィールド名の前に表示されるヘッダー文字列を明示的に指定できます。 次に例を示します。 `DISPLAY 'EMPLOYEE' NAME 'SALARY' SALARY` ■ フィールドのヘッダーを明示的に指定しない場合は、DEFINE DATA ステートメントで定義したヘッダーが使用されます。 ■ DEFINE DATA ステートメントでデータベースフィールドのヘッダーが定義されていない場合は、DDMに定義されているヘッダーがデフォルトとして使用されます。 ■ デフォルトのヘッダーがDDMに定義されていない場合は、フィールド名がヘッダーとして使用されます。 ■ DEFINE DATA ステートメントでユーザー定義変数のヘッダーが定義されていない場合は、変数名がヘッダーとして使用されます。 ヘッダー定義については、DEFINE DATA ステートメントも参照してください。 `DISPLAY NAME SALARY #NEW-SALARY` ■ Naturalでは、常に列ヘッダーに下線が付けられ、表示するデータと下線の間に 1 行の空白行が生成されます。 ■ プログラム内に複数の DISPLAY ステートメントがある場合は、最初の DISPLAY ステートメントにより、使用する列ヘッダーが決定されます。これはコンパイル時に評価されます。 列ヘッダーの省略：

	1 つのフィールドの列ヘッダーを省略するには ■ フィールド名の前に、次の文字列（アポストロフィ-スラッシュ-アポストロフィ）を指定します。 `'/'` 例えば、次のようになります。 `DISPLAY '/' NAME 'SALARY' SALARY` すべての列ヘッダーを省略するには ■ キーワード NOHDR を指定します。 `DISPLAY NOHDR NAME SALARY` **注意:** 1. NOHDR は最初の DISPLAY ステートメントにのみ有効です。これは、後続の DISPLAY ステートメントでは列ヘッダーを作成できないためです。 2. NOTITLE と NOHDR の両方を使用する場合は、DISPLAY NOTITLE NOHDR NAME SALARY の順で指定する必要があります。
GIVE SYSTEM FUNCTIONS	システム関数の使用： GIVE SYSTEM FUNCTIONS 節を使用すると、Natural システム関数 AVER、COUNT、MAX、MIN、NAVER、NCOUNT、NMIN、SUM、TOTAL が利用できるようになります。 これらのシステム関数は、GIVE SYSTEM FUNCTIONS 節を含む DISPLAY ステートメントの実行時に評価されます。 これらのシステム関数は、AT END OF PAGE 条件で実行されるステートメントで参照できます。 **注意:** 1. レポートごとに、1 つの DISPLAY ステートメントにのみ GIVE SYSTEM FUNCTIONS 節を含めることができます。 DISPLAY ステートメントからシステム関数が評価されるときに、システム関数はページ単位で評価されます。つまり、新しいページが始まると、TOTAL 以外の関数はすべてゼロにリセットされます。 2. システム関数をサブルーチン内の DISPLAY ステートメントで使用する場合は、END OF PAGE 処理を同じルーチン内で開始する必要があります。 「*例 2 - DISPLAY ステートでの GIVE SYSTEM FUNCTIONS 節の使用例*」も参照してください。
statement-parameters	ステートメントレベルでのパラメータ定義：

	1 つまたは複数のパラメータをカッコで囲んで、要素（フィールド）レベルで、つまり DISPLAY の直後に指定できます。 この方法で指定した各パラメータによって、以前に GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードのみ）、または FORMAT ステートメントで指定した対応するパラメータは上書されます。 複数のパラメータを指定する場合は、個々のパラメータを1つ以上の空白で区切る必要があります。 各パラメータ指定を 2 行のステートメント行に分割することはできません。 **注意:** ここで適用されるパラメータ設定は、変数フィールドにのみ関連し、テキスト定数には影響しません。 テキスト定数にフィールド属性を設定する場合は、この要素に属性を明示的に設定する必要があります（「*要素（フィールド）レベルでのパラメータ定義*」を参照）。 以下の項目も参照してください。 ■ パラメータのリスト ■ ステートメントおよび要素（フィールド）レベルでのパラメータ使用例 ■ 例7-ステートメント／要素レベルでパラメータを使用した**DISPLAY**ステートメント

パラメータのリスト

DISPLAY ステートメントで指定可能なパラメータ		**指定（S = ステートメントレベル、E = 要素レベル）**
AD	属性定義	SE
AL	出力の英数字長	SE
CD	カラー定義	SE
CV	制御変数	SE
DF	日付フォーマット	SE
DL	出力の表示長	SE
DY	ダイナミック属性	SE
EM	編集マスク	SE
EMU	Unicode 編集マスク	E
ES	空行の省略	S
FC	充填文字	SE
FL	浮動小数点仮数長	SE
GC	グループヘッダーの充填（フィラー）文字	SE
HC	ヘッダーの中央揃え	SE
HW	ヘッダーの幅	SE

DISPLAY ステートメントで指定可能なパラメータ		指定（S = ステートメントレベル、E = 要素レベル）
IC	挿入文字	SE
IS	重複抑制	SE
LC	先頭文字	SE
LS	行サイズ	S
MC	マルチプルバリューフィールド数	S
MP	レポートの最大ページ数	S
NL	出力の数値長	SE
PC	ピリオディックグループ数	S
PM	出力モード	SE
PS	ページサイズ	S
SF	フィールド間の空白	SE
SG	符号の位置	SE
TC	末尾文字	SE
UC	下線付き文字	SE
ZP	ゼロ出力	SE

各パラメータの詳細については、『*パラメータリファレンス*』（セッションパラメータ）を参照してください。

『*プログラミングガイド*』の次のトピックも参照してください。

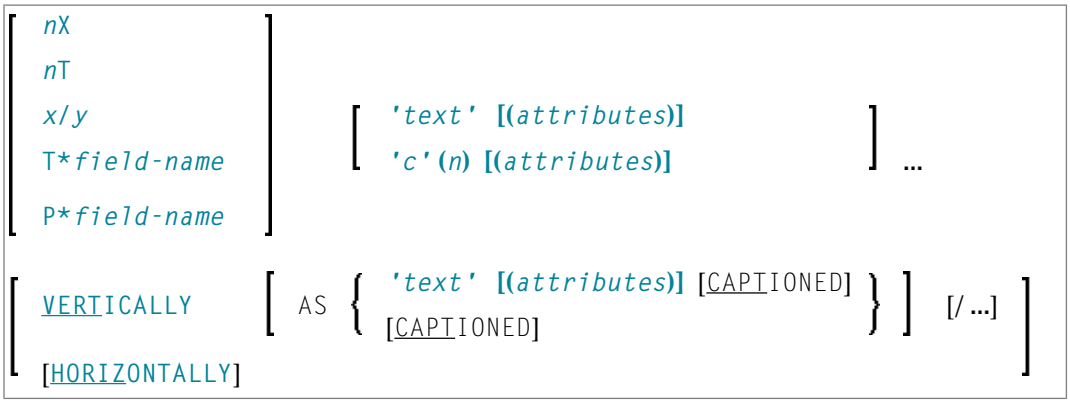
- *列ヘッダーの中央揃え - HC パラメータ*
- *列ヘッダーの幅 - HW パラメータ*
- *ヘッダーの充填文字 - FC および GC パラメータ*
- *タイトルおよびヘッダーの下線付き文字 - UC パラメータ*

ステートメントおよび要素（フィールド）レベルでのパラメータ使用例

```
DEFINE DATA LOCAL
1 VARI (A4)     INIT <'1234'>                                  /*     Output
END-DEFINE                                                     /*    Produced
*                                                              /*    ---------
DISPLAY NOHDR           'Text'          '='   VARI             /*    Text 1234
DISPLAY NOHDR (AD=U)    'Text'          '='   VARI             /*    Text 1234
DISPLAY NOHDR           'Text' (AD=U)   '='   VARI (AD=U)      /*    Text 1234
```

```
DISPLAY NOHDR           'Text' (AD=U)  '='   VARI          /*    Text 1234
END
```

出力フォーマット定義



フィールドの位置指定表記

***n*X**	列の間隔： この表記により、列の間に *n* 個のスペースが挿入されます。 例： `DISPLAY NAME 5X SALARY` 以下の項目も参照してください。 ■ *例 1 - nX および nT 表記を使用した DISPLAY ステートメント*（後述） ■ 列の間隔 - SF パラメータと *n*X 表記（『*プログラミングガイド*』）
***n*T**	タブ設定： *n*T 表記により、位置 *n* を表示するように位置指定（タブ設定）されます。 後方への位置指定はできません。 次の例では、NAME が 25 桁目から、SALARY が 50 桁目から表示されます。 `DISPLAY 25T NAME 50T SALARY` 以下の項目も参照してください。 ■ *例 1 - nX および nT 表記を使用した DISPLAY ステートメント*（後述） ■ *タブ設定 - nT 表記*（『*プログラミングガイド*』）

x/*y*	**x/y** 位置指定： *x*/*y* 表記により、次の要素は最後のステートメント出力の *x* 行下、列 *y* の先頭に配置されます。 *y* を "0" にすることはできません。 後方への位置指定はできません。
T*field-name*	フィールドの相対位置指定： T* 表記により、前の DISPLAY ステートメントで使用されたフィールドの特定の表示桁を位置指定できます。 後方への位置指定はできません。
P*field-name*	フィールドおよび行の相対位置指定： P* 表記により、前の DISPLAY ステートメントで使用されたフィールドの特定の表示桁を位置指定できます。 縦（VERT）方向の表示モードと組み合せて使用します。 後方への位置指定はできません。 以下の項目も参照してください。 ■ *例 3 - P* 表記を使用した DISPLAY ステートメント*（後述） ■ *タブ表記 P*field*（『プログラミングガイド』）

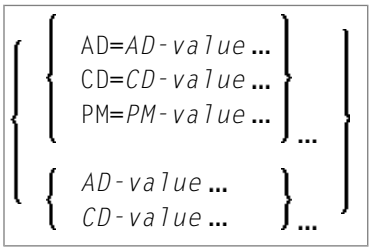
列ヘッダー割り当ての上書き

'*text*' '/'	テキスト割り当て： フィールドの直前に指定すると、一重引用符で囲まれた文字列によって列ヘッダーが上書きされます。 フィールド名の前にスラッシュ文字 '/' を指定すると、列ヘッダーは省略されます。 `DISPLAY 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT` 複数の '*text*' 要素をフィールド名の前に指定すると、*最後の* '*text*' 要素が列ヘッダーとして使用され、残りの要素は列内でフィールド値の前に配置されます。 以下の項目も参照してください。 ■ *独自の列ヘッダーの定義*（『プログラミングガイド』） ■ 「*テキスト表記*」、「*ステートメントで使用するテキストの定義*」（『プログラミングガイド』） ■ *例 4 - 'text'、'c(n)'、および属性表記を使用した DISPLAY ステートメント*（後述）
'c'(n)	文字の繰り返し：

	フィールド値の直前に、一重引用符で囲まれた文字列が*n*回表示されます。次に例を示します。 `DISPLAY '*' (5) '=' NAME` 結果 `***** SMITH` 以下の項目も参照してください。 ■ *テキスト表記、フィールド値の前にn回表示される文字の定義*（『*プログラミングガイド*』） ■ *例 4 - 'text'、'c(n)'、および属性表記を使用した* ***DISPLAY*** *ステートメント*（後述）

出力属性

attributes は、テキスト表示に使用される出力属性を示します。 可能な属性：



指定可能なセッションパラメータ値については、『*パラメータリファレンス*』ドキュメントの該当するセクションを参照してください。

■ 「*AD - 属性定義*」の「*フィールド表現*」

■ *CD - カラー定義*

■ *PM - 出力モード*

Note: コンパイラは、実際には1つの出力フィールドに複数の属性値を受け入れます。 例えば、「AD=BDI」と指定できます。 ただし、この場合は最後の値のみが適用されます。 示した例では、値 "I" のみが有効になり、出力フィールドは強調表示されます。

VERTICAL（縦）／**HORIZONTAL**（横）表示

VERT 節は、複数のフィールド値を同一の列内で上下に表示するために使用します。 VERT モードでは、キーワード VERT または HORIZ を指定すると新しい列が開始されます。

VERT モードの列ヘッダーは、次に示すように AS 節で指定したエントリで制御されます。

VERTICALLY	縦方向の列。 AS 節を省略すると、列ヘッダーは出力されません。 `DISPLAY VERT NAME SALARY` 表示例については、『*プログラミングガイド*』の「*AS 節のない DISPLAY VERT*」を参照してください。
AS '*text***'**	縦方向の列。 AS '*text*' を指定すると、文字列が列ヘッダーとして使用されます。 表示例については、『*プログラミングガイド*』の「*DISPLAY VERT AS 'text'*」を参照してください。 '*text*' の文字列中にスラッシュ文字/がある場合、列ヘッダーは改行されて複数行になります。 `DISPLAY VERT AS 'LAST/NAME' NAME`
AS '*text***' CAPTIONED**	縦方向の列。 AS '*text*' CAPTIONED を指定すると、'*text*' が列ヘッダーとして使用され、各明細行のフィールド値の直前に標準ヘッダーまたはフィールド名が挿入されます。 `DISPLAY VERT AS 'PERSONS/SELECTED' CAPTIONED NAME FIRST-NAME` 表示例については、『*プログラミングガイド*』の「*DISPLAY VERT AS 'text' CAPTIONED*」を参照してください。
AS CAPTIONED	縦方向の列。 AS CAPTIONED を指定すると、フィールドの標準ヘッダー（ヘッダー文字列あるいはフィールド名）が列ヘッダーとして使用されます。 `DISPLAY VERT AS CAPTIONED NAME FIRST-NAME`
HORIZONTALLY	横方向の列。 これはデフォルトの表示モードです。

キーワードを使用して縦方向の表示と横方向の表示を混在させることができます。

ある出力項目の縦方向の表示を一時的に停止する必要がある場合は、その要素の前に "-"（ハイフン）を指定します。 次に例を示します。

```
DISPLAY VERT NAME - FIRST-NAME SALARY
```

この例では、FIRST-NAME は NAME の横に横方向で表示され、SALARY は NAME の下に縦方向で表示されます。

標準の表示モードは、横方向です。 表示されるフィールドごとに 1 つの列が生成されます。

列ヘッダーは、Natural によって次の優先順位で取得および使用されます。

1. DISPLAY ステートメントで指定したヘッダーの '*text*'。
2. データベースフィールドについては DDM に定義されたデフォルトのヘッダー、またはユーザー定義変数については変数名。
3. データベースフィールドに対してヘッダー文字列が定義されていない場合は、DDMに定義されたフィールド名。

グループ名については、グループ全体に 1 つのグループヘッダーがつけられます。 グループを指定する場合、グループ全体に対するヘッダーだけをユーザーが変更できます。

列ヘッダーの最大行数は 15 行です。

DISPLAY ステートメントからの出力は、行の幅（ラインサイズ）を超えないようにする必要があります。 行の幅を超えると、エラーメッセージが発行されます。

縦方向の表示と横方向の表示の詳細な使用方法については、次の例を参照してください。

- *例 5 - 横方向の表示を使用した DISPLAY ステートメント*
- *例 6 - 縦および横方向の表示を使用した DISPLAY ステートメント*
- *DISPLAY VERT AS CAPTIONED および HORIZ*（『*プログラミングガイド*』）

出力項目

```
[ { 'text' [(attributes)]   }...   ['='] {operand1 [(parameters)]}
  { 'c'(n) [(attributes)]   }
    nX
    nT
    x/y                    ]
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット												参照	ダイナミック定義
		S	A	G	N	A	N	P	I	F	B	D	T	L		G	O		
operand1																		可	不可

構文要素の説明

*n***X**	列の間隔： これは、「*出力フォーマット定義*」の場合と同じです（上記参照）。
*n***T**	タブ設定： これは、「*出力フォーマット定義*」の場合と同じです（上記参照）。
x/*y*	**x/y** 位置指定： これは、「*出力フォーマット定義*」の場合と同じです（上記参照）。
'*text*'	テキスト割り当て： これは、「*出力フォーマット定義*」の場合と同じです（上記参照）。
'*c*'*(n)*	文字の繰り返し： これは、「*出力フォーマット定義*」の場合と同じです（上記参照）。
'*text*'**'='** '*c*'*(n)*'**='**	'*text*' '=' をフィールドの直前に置くと、'*text*' がフィールド値の直前に出力されます。 これは、'*c*' *(n)* '=' と同じように適用できます。 `DISPLAY '*****' '=' NAME`
attributes	出力属性： これは、「*出力フォーマット定義*」の場合と同じです（上記参照）。
operand1	表示フィールド。
parameters	要素（フィールド）レベルでのパラメータ定義： 1 つまたは複数のパラメータをカッコで囲んで、要素（フィールド）レベルで、つまり *operand1* の直後に指定できます。 この方法で指定した各パラメータは、以前にステートメントレベル、GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードでのみ）、または FORMAT ステートメントで指定した対応するパラメータを上書きします。 複数のパラメータを指定する場合は、各エントリ間に 1 つ以上の空白を配置する必要があります。 エントリを 2 行のステートメント行に分割することはできません。 以下の項目も参照してください。 ■ *パラメータのリスト* ■ *ステートメントおよび要素（フィールド）レベルでのパラメータ使用例*

DISPLAY ステートメントに適用されるデフォルト

DISPLAY ステートメントには、次のデフォルトが適用されます。

- レポートの幅
  レポートの幅は、Naturalをインストールしたときの設定値がデフォルトになります。このデフォルト値は通常、バッチモードでは 132 であり、TP モードでは端末の 1 行の長さです。 この値は、セッションパラメータ LS で上書きできます。TP モードでは、行の長さ（LS）とページの大きさ（PS）のパラメータは、使用中の端末タイプの物理特性に基づいて Natural で設定されます。
- 端末画面の出力
  DISPLAY ステートメントの出力を端末画面に表示する場合、出力は物理的に 2 桁目から始まります。これは、1 桁目が 3270 タイプ端末の属性桁として予約されているためです。
- 用紙への印刷
  DISPLAY ステートメントの出力を用紙に印刷する場合は、1 桁目から印刷されます。
- フィールド間の空白
  出力項目間のデフォルトのフィールド間の空白は 1 桁です。 したがって、列の間には最小 1 桁の空白があけられます（端末属性用に予約済）。 この値は、セッションパラメータ SF で上書きできます。
- フィールド出力
  フィールド長、またはフィールドヘッダーのうちいずれか長い方によって、レポートの列の幅が決まります（HW パラメータが使用されない場合）。
  - フィールドの方がヘッダーよりも長い場合、パラメータ HC=L または HC=R でヘッダーを左寄せまたは右寄せにする指示がない限り、ヘッダーは列の上でセンタリングされます。
  - また、ヘッダーの方がフィールドより長い場合、フィールドはヘッダーの下に左詰めで出力されます。
  - フィールド内の値は、英数字フィールドでは左詰めになり、数字フィールドでは右詰めになります。
  - 数値フィールドは、AD=L を指定すると左詰めで表示されます。
  - 英数字フィールドは、AD=R を指定すると右詰めで表示できます。
  - 縦方向の表示では、フィールドの中で最も長いデータ値またはヘッダーによって列の幅が決まります（HW パラメータを使用しない場合）。
- 符号
  数字フィールドの印刷時には、符号桁として 1 桁余分に左側に確保されます。 Natural セッションパラメータ SG を使用すると、符号桁を省略できます。

■ ページオーバーフロー
DISPLAY ステートメントの実行前に、ページオーバーフローがチェックされます。DISPLAY ステートメントの実行中に、新しいページタイトルまたはトレイラ情報が生成されることはありません。

例

例 1 - *n*X および *n*T 表記を使用した DISPLAY ステートメント

```
** Example 'DISEX1': DISPLAY (with nX, nT notation)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY NAME
  DISPLAY NOTITLE 5X NAME 50T JOB-TITLE
END-READ
*
END
```

プログラム **DISEX1** の出力：

```
       NAME                                        CURRENT
                                                  POSITION
--------------------                          -------------------------

ABELLAN                                       MAQUINISTA
ACHIESON                                      DATA BASE ADMINISTRATOR
ADAM                                          CHEF DE SERVICE
ADKINSON                                      PROGRAMMER
```

例 2 - DISPLAY ステートでの GIVE SYSTEM FUNCTIONS 節の使用例

```
** Example 'DISEX2': DISPLAY (with GIVE SYSTEM FUNCTIONS)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 15
FORMAT PS=15
*
READ EMPLOY-VIEW
  DISPLAY GIVE SYSTEM FUNCTIONS
          PERSONNEL-ID NAME FIRST-NAME SALARY (1) CURR-CODE (1)
  AT END OF PAGE
    WRITE /    'SALARY STATISTICS:'
          / 7X 'MAXIMUM:' MAX(SALARY(1))  CURR-CODE (1)
          / 7X 'MINIMUM:' MIN(SALARY(1))  CURR-CODE (1)
          / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDPAGE
END-READ
*
END
```

プログラム **DISEX2** の出力：

```
Page      1                                              05-01-12  09:47:48

PERSONNEL         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     CODE
--------- -------------------- -------------------- ---------- --------

50005500  BLOND               ALEXANDRE               172000 FRA
50005300  MAIZIERE            ELISABETH               166900 FRA
```

```
50004900  CAOUDAL              ALBERT                    167350 FRA
50004600  VERDIE               BERNARD                   170100 FRA
50004200  VAUZELLE             BERNARD                   159790 FRA
50004100  CHAPUIS              ROBERT                    169900 FRA
50003800  JOUSSELIN            DANIEL                    171990 FRA
50006900  BAILLET              PATRICK                   188000 FRA
50007600  MARX                 JEAN-MARIE                365700 FRA

SALARY STATISTICS:
       MAXIMUM:     365700 FRA
       MINIMUM:     159790 FRA
       AVERAGE:     192414 FRA
```

例 3 - P* 表記を使用した DISPLAY ステートメント

```
** Example 'DISEX3': DISPLAY (with P* notation)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 SALARY (1)
  2 BIRTH
  2 CITY
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY CITY FROM 'N'
 DISPLAY NOTITLE NAME CITY
          VERT AS 'BIRTH/SALARY' BIRTH (EM=YYYY-MM-DD) SALARY (1)
  SKIP 1
  AT BREAK OF CITY
    DISPLAY P*SALARY (1) AVER(SALARY (1))
    SKIP 1
  END-BREAK
END-READ
END
```

プログラム **DISEX3** の出力：

```
        NAME                CITY           BIRTH
                                           SALARY
-------------------- -------------------- ----------

WILCOX               NASHVILLE            1970-01-01
                                               38000

MORRISON             NASHVILLE            1949-07-10
                                               36000
```

```
                                                37000
```

例 4 - '*text*'、'c(*n*)'、および属性表記を使用した DISPLAY ステートメント

```
** Example 'DISEX4': DISPLAY (with 'c(n)' notation and attribute)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 DEPT
  2 LEAVE-DUE
  2 NAME
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY DEPT FROM 'T'
  IF LEAVE-DUE GT 40
   DISPLAY NOTITLE
           'EMPLOYEE' NAME                /* OVERRIDE STANDARD HEADER
           'LEAVE ACCUMULATED' LEAVE-DUE  /* OVERRIDE STANDARD HEADER
           '*'(10)(I)                     /* DISPLAY 10 '*' INTENSIFIED

  ELSE
    DISPLAY NAME LEAVE-DUE
  END-IF
END-READ
*
END
```

プログラム **DISEX4** の出力：

```
      EMPLOYEE       LEAVE ACCUMULATED
-------------------- -----------------

LAVENDA               33
BOYER                 33
```

```
CORREARD              45               **********
BOUVIER               19
```

例 5 - 横方向の表示を使用した DISPLAY ステートメント

```
** Example 'DISEX5': DISPLAY (horizontal display)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 SALARY    (1:2)
  2 CURR-CODE (1:2)
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY NAME
 DISPLAY NOTITLE NAME JOB-TITLE SALARY (1:2) CURR-CODE (1:2)
  SKIP 1
END-READ
*
END
```

プログラム **DISEX5** の出力：

```
        NAME                  CURRENT            ANNUAL   CURRENCY
                             POSITION            SALARY     CODE
-------------------- ------------------------- ---------- --------

ABELLAN              MAQUINISTA                  1450000 PTA
                                                 1392000 PTA

ACHIESON             DATA BASE ADMINISTRATOR       11300 UKL
                                                   10500 UKL

ADAM                 CHEF DE SERVICE              159980 FRA
                                                       0
```

```
ADKINSON             PROGRAMMER                      34500 USD
                                                     31700 USD
```

例 6 - 縦および横方向の表示を使用した DISPLAY ステートメント

```
** Example 'DISEX6': DISPLAY (vertical and horizontal display)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 JOB-TITLE
  2 SALARY    (1:2)
  2 CURR-CODE (1:2)
END-DEFINE
*
LIMIT 1
READ EMPL-VIEW BY NAME
 DISPLAY NOTITLE VERT AS CAPTIONED
          NAME CITY 'POSITION' JOB-TITLE
          HORIZ 'SALARY' SALARY (1:2) 'CURRENCY' CURR-CODE (1:2)
  /*
  SKIP 1
END-READ
END
```

プログラム **DISEX6** の出力：

```
          NAME              SALARY   CURRENCY
          CITY
        POSITION
------------------------ ---------- --------

ABELLAN                     1450000 PTA
MADRID                      1392000 PTA
MAQUINISTA
```

例 7 - ステートメント／要素レベルでパラメータを使用した DISPLAY ステートメント

```
** Example 'DISEX7': DISPLAY (with parameters for statement/element)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
  2 TELEPHONE
    3 AREA-CODE
```

```
    3 PHONE
END-DEFINE
*
LIMIT 3
READ EMPL-VIEW BY NAME
 DISPLAY NOTITLE (AL=16 GC=+ NL=8 SF=3 UC==)
          PERSONNEL-ID NAME TELEPHONE (LC=< TC=>)
END-READ
END
```

プログラム **DISEX7** の出力：

```
   PERSONNEL             NAME         +++++++++++++++TELEPHONE+++++++++++++++
       ID
                                             AREA              TELEPHONE
                                             CODE
================   ================   ==================   ==================

60008339           ABELLAN            <1     >             <4356726        >
30000231           ACHIESON           <0332  >             <523341         >
50005800           ADAM               <1033  >             <44864858       >
```

例 8 - 出力ファイルを PC として Natural に定義する場合のレポート指定

```
** Example 'PCDIEX1': DISPLAY and WRITE to PC
**
** NOTE: Example requires that Natural Connection is installed.
************************************************************************
DEFINE DATA LOCAL
01 PERS VIEW OF EMPLOYEES
  02 PERSONNEL-ID
  02 NAME
  02 CITY
END-DEFINE
*
FIND PERS WITH CITY = 'NEW YORK'               /* Data selection
  WRITE (7) TITLE LEFT 'List of employees in New York' /
  DISPLAY (7)          /* (7) designates the output file (here the PC).
    'Location'  CITY
    'Surname'   NAME
    'ID'        PERSONNEL-ID
END-FIND
END
```

55 DIVIDE

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ADD** | **COMPRESS** | **COMPUTE** | **EXAMINE** | **MOVE** | **MOVE ALL** | **MULTIPLY** | **RESET** | **SEPARATE** | **SUBTRACT**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

DIVIDE ステートメントは、2 つのオペランドを除算するために使用します。

Note: ゼロによる除算に関する注意事項："0" になっている除数（*operand1*）を使用しようとすると、エラーメッセージまたは結果として "0" が返されます。いずれが返されるかは、セッションパラメータ ZD の設定によって決まります（詳細については、『パラメータリファレンス』ドキュメントを参照）。

構文説明

このステートメントには異なる構造が可能です。

- **構文 1 - DIVIDE（GIVING 節を含まない場合）**
- **構文 2 - DIVIDE（GIVING 節を含む場合）**
- **構文 3 - DIVIDE（REMAINDER オプションを含む場合）**

構文 1 - DIVIDE（GIVING 節を含まない場合）

```
DIVIDE [ROUNDED] operand1 INTO operand2
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

オペランド定義テーブル：

オペランド	構文要素			フォーマット	オペランド参照	ダイナミック定義
operand1	C	S	A	N N P I F	可	不可
operand2	C	S	A	M N P I F	可	不可

構文要素の説明：

operand1 **INTO** *operand2*	オペランド： *operand1*は除数で、*operand2*は被除数です。 結果は*operand2*（結果フィールド）に格納されるため、ステートメントは次の式と同等になります。 `<oper2> := <oper2> / <oper1>` 結果フィールドとして使用できるのは、データベースフィールドまたはユーザー定義変数です。 *operand2*が定数または変更不可のNaturalシステム変数の場合は、`GIVING`節が必要です。 除算の結果の小数点位置は、結果フィールド（つまり、*operand2*）から計算されます。
ROUNDED	キーワード `ROUNDED` を指定すると、結果は切り上げられます。

構文 2 - DIVIDE（GIVING 節を含む場合）

```
DIVIDE [ROUNDED] operand1 INTO operand2  [GIVING operand3]
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

オペランド定義テーブル：

オペランド	構文要素					フォーマット												オペランド参照	ダイナミック定義
operand1	C	S	A		N			N	P	I	F							可	不可
operand2	C	S	A		N			N	P	I	F							可	不可
operand3		S	A			A	U	N	P	I	F	B*						可	可

**operand3*のフォーマットBは、4以下の長さでのみ使用できます。

構文要素の説明：

operand1 **INTO** *operand2* **GIVING** *operand3*	オペランド：

	operand1 は除数で、*operand2* は被除数です。結果は *operand3* に格納されるため、ステートメントは次の式と同等になります。 `<oper3> := <oper2> / <oper1>` 結果フィールドとしてデータベースフィールドを使用している場合は、除算の結果により更新されるのはプログラム内で使用されるフィールドの内部値のみです。データベース内のフィールドの値は変更されないまま維持されます。 除算の結果の小数点位置は、結果フィールド（つまり、*operand3*）から計算されます。 結果の精度については、『*プログラミングガイド*』の「*演算割り当てのルール*」、「*算術演算結果の精度*」を参照してください。
ROUNDED	キーワード ROUNDED を指定すると、結果は切り上げられます。

構文 3 - DIVIDE（REMAINDER オプションを含む場合）

```
DIVIDE operand1  INTO operand2 [GIVING operand3] REMAINDER operand4
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

オペランド定義テーブル：

オペランド	構文要素			フォーマット	オペランド参照	ダイナミック定義
operand1	C	S	A	N N P I	可	不可
operand2	C	S	A	N N P I	可	不可
operand3		S	A	A U N P I F B*	可	可
operand4		S	A	A U N P I F B* T	可	可

* *operand3* および *operand4* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

operand1	除数： *operand1* は除数、つまり割り算の商を求めるために被除数を割る数値または量のことです。
operand2	結果フィールド： GIVING 節を使用しない場合は、*operand2* に結果が格納されます。 結果フィールドとして使用できるのは、データベースフィールドまたはユーザー定義変数です。 *operand2* が定数または変更不可の Natural システム変数の場合は、GIVING 節が必要です。

ROUNDED	キーワード ROUNDED を指定すると、結果は切り上げられます。
GIVING *operand3*	キーワード GIVING を使用すると、*operand2* は変更されず、*operand3* に結果が格納されます。 結果フィールドとしてデータベースフィールドを使用している場合は、除算の結果により更新されるのはプログラム内で使用されるフィールドの内部値のみです。 データベース内のフィールドの値は変更されないまま維持されます。 除算結果の小数桁数は、結果フィールド（つまり GIVING 節を使用している場合は *operand2*、GIVING 節を使用していない場合は *operand3*）で評価されます。 結果の精度については、『*プログラミングガイド*』の「*演算割り当てのルール*」、「*算術演算結果の精度*」を参照してください。
REMAINDER *operand4*	キーワード REMAINDER を指定すると、除算の余りが指定フィールド（*operand4*）に挿入されます。 GIVING と REMAINDER を併用する場合、4 つのオペランドに配列範囲を使用することはできません。 余りは、内部的に次のように計算されます。 1. *operand1* と *operand2* の除算の商が計算されます。 2. 商は *operand1* と掛け合わされます。 3. この乗算の結果が *operand2* から引かれます。 4. この減算の結果は、*operand4* に割り当てられます。 これらの各ステップにおいて、「*算術演算結果の精度*」（『*プログラミングガイド*』）で説明されている規則が適用されます。

例

```
** Example 'DIVEX1': DIVIDE
************************************************************************
DEFINE DATA LOCAL
1 #A (N7) INIT <20>
1 #B (N7)
1 #C (N3.2)
1 #D (N1)
1 #E (N1) INIT <3>
1 #F (N1)
END-DEFINE
*
DIVIDE 5 INTO #A
WRITE NOTITLE 'DIVIDE 5 INTO #A' 20X '=' #A
*
RESET INITIAL #A
```

```
DIVIDE 5 INTO #A GIVING #B
WRITE 'DIVIDE 5 INTO #A GIVING #B' 10X '=' #B
*
DIVIDE 3 INTO 3.10 GIVING #C
WRITE 'DIVIDE 3 INTO 3.10 GIVING #C' 8X '=' #C
*
DIVIDE 3 INTO 3.1 GIVING #D
WRITE 'DIVIDE 3 INTO 3.1 GIVING #D' 9X '=' #D
*
DIVIDE 2 INTO #E REMAINDER #F
WRITE 'DIVIDE 2 INTO #E REMAINDER #F' 7X '=' #E '=' #F
*
END
```

プログラム **DIVEX1** の出力：

```

DIVIDE 5 INTO #A                    #A:        4
DIVIDE 5 INTO #A GIVING #B          #B:        4
DIVIDE 3 INTO 3.10 GIVING #C        #C:    1.03
DIVIDE 3 INTO 3.1 GIVING #D         #D:  1
DIVIDE 2 INTO #E REMAINDER #F       #E:  1 #F:  1
```

56 DO/DOEND

```
DO statement ... DOEND
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

DO および DOEND ステートメントは、次のステートメントの論理条件に基づいて実行されるステートメント群を指定するために使用します。

- AT BREAK
- AT END OF DATA
- AT END OF PAGE
- AT START OF DATA
- AT TOP OF PAGE
- BEFORE BREAK PROCESSING
- FIND ... IF NO RECORDS FOUND
- IF
- IF SELECTION
- ON ERROR
- READ WORK FILE ... AT END OF FILE

制限事項

- DO および DOEND ステートメントはレポーティングモードでのみ有効です。
- WRITE TITLE、WRITE TRAILER、および AT 条件ステートメント AT BREAK、AT END OF DATA、AT END OF PAGE、AT START OF DATA、AT TOP OF PAGE は、DO/DOEND ステートメント群の*中*では指定できません。
- ループ開始ステートメントを DO/DOEND 内のステートメント群に使用する場合、そのループは DOEND の前に LOOP ステートメントで終了する必要があります。

例

```
** Example 'DOEEX1': DO/DOEND
************************************************************************
*
EMP. FIND EMPLOYEES WITH CITY = 'MILWAUKEE'
  VEH. FIND VEHICLES WITH PERSONNEL-ID = PERSONNEL-ID
    IF NO RECORDS FOUND DO
      ESCAPE
    DOEND
    DISPLAY PERSONNEL-ID (EMP.) NAME (EMP.)
            SALARY (EMP.,1)
            MAKE (VEH.) MAINT-COST (VEH.,1)
    AT END OF DATA DO
      WRITE NOTITLE
        / 10X 'AVG SALARY:'
              T*SALARY (1) AVER(SALARY (1))
        / 10X 'AVG MAINTENANCE (ZERO VALUES EXCLUDED):'
              T*MAINT-COST (1) NAVER(MAINT-COST (1))
    DOEND
    /*
  LOOP
LOOP
END
```

プログラム **DOEEX1** の出力：

```
PERSONNEL         NAME           ANNUAL           MAKE         MAINT-COST
   ID                            SALARY
--------- -------------------- ---------- -------------------- ----------

20021100  JONES                     31000 GENERAL MOTORS            140
20027800  LAWLER                    29000 GENERAL MOTORS              0
20027800  LAWLER                    29000 TOYOTA                     86
20030600  NORDYKE                   47000 FORD                      194

          AVG SALARY:               35666
          AVG MAINTENANCE (ZERO VALUES EXCLUDED):                   140
```

57 EJECT

このchapterでは、次のトピックについて説明します。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

EJECT ステートメントは、改ページとページ換えの制御に使用します。

構文説明

このステートメントには、2 つの異なる構造が可能です。

- EJECT - **構文 1**
- EJECT - **構文 2**

次の構文図で使用されている記号については、「*構文記号*」を参照してください。

EJECT - 構文 1

構文要素の説明：

EJECT ON/OFF (*rep*)	レポート指定を含む **EJECT ON/OFF -** オンラインおよびバッチモード：	
	EJECT OFF (*rep*)	指定レポートのページ換え処理を行いません（構文 **2** の EJECT ステートメントの指定は除く）。
	EJECT ON (*rep*)	指定レポートのページ換え処理を実行します。
EJECT ON/OFF	レポート指定を含まない **EJECT ON/OFF -** バッチモードのみ： レポート指定 (*rep*) のない EJECT ON/OFF では、バッチモードで使用して、プログラムの実行中に作成される出力リスト間のページ換え処理を制御できます。	
	EJECT ON	ソースプログラムリスト、出力レポートおよび実行完了メッセージ「EXECUTION COMPLETED」 の間でページ換えを実行します。 これはデフォルト設定です。

	EJECT OFF	上記の各出力間のページ換えを停止します。EJECT OFF は次の EJECT ON ステートメントが指定されるまで有効です。
(*rep*)	レポート指定： 表記（*rep*）は、EJECT ステートメントを適用するレポートの ID を指定するために使用できます。 範囲 0～31 の値、または DEFINE PRINTER ステートメントを使用して割り当てた論理名を指定できます。 "(*rep*)" を指定しない場合、EJECT ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。	

EJECT - 構文 2

この形式の EJECT ステートメントでは、次のページでタイトルまたはヘッダー行を生成することなく、TOP/END PAGE 処理を行わずに改ページが実行されます。

```
EJECT  [( rep )]  [ [ IF ]  LESS [THAN] operand1  [LINES] [LEFT]  ]
                  [ [WHEN]                                        ]
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット										オペランド参照	ダイナミック定義
operand1	C	S				N	P	I								可	不可

構文要素の説明：

(*rep*)	レポート指定： 表記（*rep*）は、EJECT ステートメントを適用するレポートの ID を指定するために使用できます。 範囲 0～31 の値、または DEFINE PRINTER ステートメントを使用して割り当てた論理名を指定できます。 （*rep*）を指定しない場合、EJECT ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。

IF LESS THAN *operand1* **LINES LEFT**	現在の行の位置が、ページサイズから *operand1* の値を引いた値よりも大きいときに限り、ページ換えが行われます。 *operand1* の値は数値定数または変数で指定できます。

処理

EJECT ステートメントが実行されても、AT TOP OF PAGE、AT END OF PAGE、WRITE TITLE、または WRITE TRAILER ステートメントが実行されることはありません。 また、DISPLAY GIVE SYSTEM FUNCTIONS で評価されるシステム関数にも影響しません。

EJECT では、新規の物理ページの生成だけが行われます。 Natural システム変数 *LINE-COUNT が "1" に設定されますが、*PAGE-NUMBER は影響を受けません。

例

```
** Example 'EJTEX1': EJECT
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
FORMAT PS=15
LIMIT 9
READ EMPLOY-VIEW BY CITY
  /*
  AT START OF DATA
    EJECT
    WRITE /// 20T '%' (29) /
              20T '%%'                            47T '%%' /
              20T '%%' 3X 'REPORT OF EMPLOYEES' 47T '%%' /
              20T '%%' 3X '  SORTED BY CITY   ' 47T '%%' /
              20T '%%'                            47T '%%' /
              20T '%' (29) /
    EJECT
  END-START
  EJECT WHEN LESS THAN 3 LINES LEFT
  /*
  WRITE '*' (64)
  DISPLAY NOTITLE NOHDR CITY NAME JOB-TITLE 5X *LINE-COUNT
  WRITE '*' (64)
```

```
END-READ
END
```

プログラム **EJTEX1** の出力：

```
                          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                          %%                          %%
                          %%   REPORT OF EMPLOYEES    %%
                          %%     SORTED BY CITY       %%
                          %%                          %%
                          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

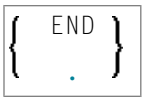
Enter キーを押した後：

```
************************************************************************
AIKEN                SENKO                PROGRAMMER                    2
************************************************************************
************************************************************************
AIX EN OTHE          GODEFROY             COMPTABLE                     5
************************************************************************
************************************************************************
AJACCIO              CANALE               CONSULTANT                    8
************************************************************************
************************************************************************
ALBERTSLUND          PLOUG                KONTORASSISTENT              11
************************************************************************
************************************************************************
ALBUQUERQUE          HAMMOND              SECRETARY                    14
************************************************************************
```

Enter キーを押した後：

```
************************************************************************
ALBUQUERQUE          ROLLING              MANAGER                       2
************************************************************************
************************************************************************
ALBUQUERQUE          FREEMAN              MANAGER                       5
************************************************************************
************************************************************************
ALBUQUERQUE          LINCOLN              ANALYST                       8
************************************************************************
************************************************************************
ALFRETON             GOLDBERG             JUNIOR                       11
************************************************************************
```

58 END



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

END ステートメントは、Natural プログラムの物理的な終わりをマークするために使用します。END ステートメントの後に記号を挿入することはできません。

レポーティングモードでは、現在アクティブになっているすべての処理ループ（LOOP ステートメントで終了していないもの）が END ステートメントで終了します。

プログラム実行に関する考慮事項

メインプログラム（レベル 1 で実行中のプログラム）で END ステートメントに到達すると、参照指定（*r*）で処理ループに関連付けられていないユーザー開始ブレイク（PERFORM BREAK PROCESSING）の最終ブレイク処理と、最後の END PAGE 処理が行われます。

サブプログラムまたは FETCH RETURN ステートメントで呼び出されたプログラムで、END ステートメントに到達すると、終了処理を行わずに制御が呼び出し元のプログラムに返されます。

構文説明

END	キーワード： 通常、Natural 予約キーワード END ステートメントは、Natural プログラムの物理的な終わりをマークするために使用します。
.	ピリオド： Natural 予約キーワード END の代わりに、ピリオド（.）を使用することもできます。 同じ行に他のステートメントが含まれている場合は、ピリオドの前に少なくとも 1 つの空白を挿入する必要があります。

例

いくつかの一般的な例については、「*DEFINE DATA ステートメントの使用例*」を参照してください。

59 END TRANSACTION

```
END [OF] TRANSACTION [operand1 ...]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

END TRANSACTION ステートメントは、論理トランザクションの終了を示すために使用します。論理トランザクションは、データベースに含まれている情報の論理的一貫性を保証するために、完全に実行される必要がある最小の業務ユニットです。業務ユニットの定義はユーザーが行います。

END TRANSACTION ステートメントが正しく実行されると、後続のユーザー、Natural、データベース、またはオペレーティングシステムによる中断に関係なく、トランザクション中のすべての更新処理をデータベースに対して物理的に適用することが保証されます。END TRANSACTION ステートメントが正しく終了しなかった場合、トランザクション中の更新処理は自動的にバックアウトされます。

また、END TRANSACTION ステートメントでは、トランザクション中にホールド状態となっていたすべてのレコードが解放されます。

END TRANSACTION ステートメントは、論理条件に基づいて実行できます。

詳細については、『*プログラミングガイド*』の「*データベース更新- トランザクション処理*」セクションを参照してください。

制限事項

このステートメントを、Entire System Server で使用することはできません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット														オペランド参照	ダイナミック定義
operand1	C	S			N	A	U	N	P	I	F	B	D	T						可	不可

構文要素の説明：

operand1	トランザクションデータの保存： Adabas データベースに適用されるトランザクションに対して、ユーザーは、このステートメントを使用して、トランザクション関連情報を保存することもできます。 ただし、トランザクションデータは 2000 バイトを超えないようにする必要があります。 トランザクションデータは、GET TRANSACTION DATA ステートメントで読み込むことができます。 トランザクションデータは、Natural プロファイルパラメータ ETDB で指定したデータベースに書き込まれます。 ETDB パラメータを指定していない場合、トランザクションデータは、プロファイルパラメータ UDB で指定したデータベースに書き込まれます（メインフレーム環境を除く）。メインフレーム環境では、トランザクションデータは、Natural Security システムファイル（FSEC）が存在するデータベースに書き込まれます。FSEC を指定していない場合、Natural システムファイル（FNAT）と同じであるとみなされます。Natural Security をインストールしていない場合は、トランザクションデータは FNAT が存在するデータベースに書き込まれます。

関連データベース

トランザクションデータ（*operand1*）が*指定されていない* END TRANSACTION ステートメントは、Natural の制御下で、データベーストランザクションが行われたときにのみ実行されます。Natural プロファイルパラメータ ET の設定により、ステートメントがどのように実行されるかが決まります。ET=OFF を指定すると、トランザクションの影響を受けたデータベースに対してのみ実行されます。ET=ON を指定すると、最後の BACKOUT TRANSACTION または END TRANSACTION ステートメントの実行以後に参照された、全データベースに対して実行されます。

トランザクションデータ（*operand1*）が指定されている END TRANSACTION ステートメントは常に実行され、データベースにトランザクションデータが格納されます（下記のセクションを参照）。 また、他のどのデータベースに対して END TRANSACTION ステートメントが実行されるかは、ET パラメータの設定で決まります（上記を参照）。

データベース固有の考慮事項

SQL データベース	ほとんどのSQLデータベースはワークの論理ユニットが終了するときにすべてのカーソルを閉じるため、END TRANSACTION ステートメントをデータベースの更新処理ループ内に指定することはできません。このようなループの後に指定する必要があります。
XML データベース	END TRANSACTION ステートメントをデータベースの更新処理ループ内に指定することはできません。このようなループの後に指定する必要があります。

例

例 1 - END TRANSACTION

```
** Example 'ETREX1': END TRANSACTION
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'BOSTON'
  ASSIGN COUNTRY = 'USA'
  UPDATE
  END TRANSACTION
  /*
  AT END OF DATA
    WRITE NOTITLE *NUMBER 'RECORDS UPDATED'
  END-ENDDATA
  /*
END-FIND
END
```

プログラム **ETREX1** の出力：

```
7 RECORDS UPDATED
```

例 2 - ET データを指定した END TRANSACTION

```
** Example 'ETREX2': END TRANSACTION (with ET data)
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 CITY
*
1 #PERS-NR (A8) INIT <' '>
END-DEFINE
*
REPEAT
  INPUT 'ENTER PERSONNEL NUMBER TO BE UPDATED:' #PERS-NR
  IF #PERS-NR = ' '
    ESCAPE BOTTOM
  END-IF
  /*
  FIND EMPLOY-VIEW PERSONNEL-ID = #PERS-NR
    INPUT (AD=M)    NAME / FIRST-NAME / CITY
    UPDATE
    END TRANSACTION #PERS-NR
  END-FIND
  /*
END-REPEAT
END
```

プログラム **ETREX2** の出力：

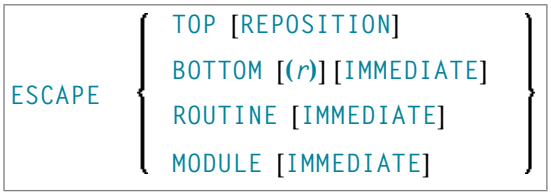
```
ENTER PERSONNEL NUMBER TO BE UPDATED: 20027800
```

社員番号を入力して確認した後：

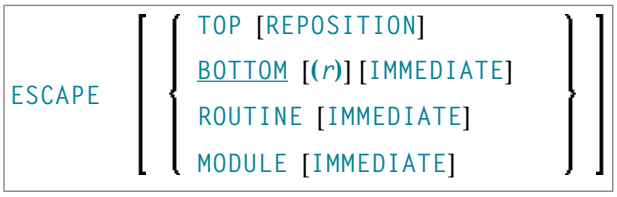
```
NAME LAWLER
FIRST-NAME SUNNY
CITY MILWAUKEE
```

60 ESCAPE

ストラクチャードモード構文



レポーティングモード構文



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：

- **FOR** | **REPEAT** | **PROCESS PAGE MODAL**
- **CALL** | **CALL FILE** | **CALL LOOP** | **CALLNAT** | **DEFINE SUBROUTINE** | **FETCH** | **PERFORM**

関連機能グループ：

- *ループ実行*
- *プログラムおよびルーチンの呼び出し*

機能

ESCAPE ステートメントは、処理ループやルーチンの実行を中断するために使用します。

キーワードの TOP、BOTTOM、および ROUTINE によって、ESCAPE ステートメントの実行後に処理をどこから続けるかを指定します。

ESCAPE TOP および ESCAPE BOTTOM ステートメントは、内部的に最も内側でアクティブになっている処理ループを参照します。 ESCAPE ステートメントを、物理的に処理ループ内に置く必要はありません。

ESCAPE TOP および ESCAPE BOTTOM ステートメントをルーチン内（サブルーチン、サブプログラム、または FETCH RETURN で起動されたプログラム）に指定すると、処理ループの実行中に呼び出されたルーチンは自動的に終了します。

その他の考慮事項

同一の処理ループ内に、複数の ESCAPE ステートメントを指定できます。

ESCAPE ステートメントは、論理条件に基づいて実行できます。 ESCAPE ステートメントが AT END OF DATA、AT BREAK、または AT END OF PAGE ブロックの処理中に実行されると、特殊条件ブロックの実行を終了し、指定された ESCAPE 処理を行います。

IF NO RECORDS FOUND 条件の処理中に ESCAPE ステートメントが実行されると、ループ終了処理は行われません（ESCAPE IMMEDIATE と同じ）。

構文説明

ESCAPE TOP	ESCAPE TOP を実行すると、処理ループの先頭から処理が続行されます。 処理ループが再び先頭から繰り返されます。
REPOSITION	ESCAPE TOP REPOSITION ステートメントを実行すると、Natural では開始値として検索変数の現在の値を使用して、即座にアクティブな READ ループの先頭から処理が続行されます。 同時に、ESCAPE TOP REPOSITION により、システム変数 *COUNTER が "0" にリセットされます。 ESCAPE TOP REPOSITION は、Adabas、DL/I、または VSAM データベースにアクセスする READ ステートメント処理ループ内で指定できます。 関連する READ ステートメントでは、WITH REPOSITION オプションを指定する必要があります。
ESCAPE BOTTOM	ESCAPE BOTTOM を実行すると、処理ループ終了後の最初のステートメントから処理が続行されます。 ループは終了し、ループ終了処理（最終の BREAK 処理、END DATA 処理）が実行されます。 レポーティングモードでは、ESCAPE BOTTOM がデフォルトです。
(*r*)	表記 (*r*)：ESCAPE BOTTOM に続けてラベルや参照番号を指定すると、ラベルや参照番号で示した処理ループの直後の最初のステートメントから処理が続行されます。
IMMEDIATE	キーワード IMMEDIATE を指定すると、ループ終了処理は行われません。
ESCAPE ROUTINE	ESCAPE ROUTINE を実行すると、PERFORM、CALLNAT、FETCH RETURN で呼び出された現在の Natural ルーチン、またはメインプログラムとしての現在の Natural ルーチンから制御が解放されます。 サブルーチンの場合、そのサブルーチンを呼び出したステートメントの、次のステートメントから処理が続行されます。 メインプログラムの場合は、Natural コマンドモードに入ります。

	ルーチン内でアクティブになっているループはすべて終了し、ループ終了処理、およびユーザー開始（PERFORM BREAK）処理に対する最終処理が行われます。 ESCAPE ROUTINE を含むプログラムをメインプログラム（レベル1）として実行している場合に、最終のEND PAGE処理が行われます。
ESCAPE MODULE	ESCAPE MODULE を実行すると、現在のプログラムレベル全体（すべての内部サブルーチンを含む）の制御が中止されます。その後、制御は前のプログラムレベルのオブジェクトに返されます。 内部サブルーチンの階層内で ESCAPE MODULE を使用すると、このレベルで動作しているすべてのルーチンを同時に終了できます。 アクティブな内部サブルーチンがない場合、ESCAPE MODULE の結果は ESCAPE ROUTINE と同じになります。 ESCAPE MODULE は、内部サブルーチンにのみに関係しています。 外部サブルーチン、サブプログラム、および呼び出されたプログラムでは、ESCAPE ROUTINE と同じ効果があります。 ESCAPE ROUTINE と同様に、ループ終了処理が実行されます。 ただし、キーワード IMMEDIATE を指定した場合、ループ終了処理は実行されません。

例

```
** Example 'ESCEX1': ESCAPE
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 FIRST-NAME
  2 NAME
  2 AREA-CODE
  2 PHONE
*
1 #CITY (A20) INIT <' '>
1 #CNTL (A1)  INIT <' '>
END-DEFINE
*
REPEAT
  INPUT 'ENTER VALUE FOR CITY:  ' #CITY
      / 'OR ''.'' TO TERMINATE  '
  IF #CITY = '.'
    ESCAPE BOTTOM
  END-IF
  /*
  FND. FIND EMPLOY-VIEW WITH CITY = #CITY
    /*
    IF NO RECORDS FOUND
      WRITE 'NO RECORDS FOUND'
      ESCAPE BOTTOM (FND.)
    END-NOREC
    AT START OF DATA
```

```
      INPUT (AD=O) 'RECORDS FOUND:' *NUMBER //
                   'ENTER ''D'' TO DISPLAY RECORDS' #CNTL (AD=M)
      IF #CNTL NE 'D'
        ESCAPE BOTTOM (FND.)
      END-IF
    END-START
    /*
    DISPLAY NOTITLE NAME FIRST-NAME PHONE
  END-FIND
END-REPEAT
```

プログラム **ESCEX1** の出力：

```
ENTER VALUE FOR CITY:  PARIS
(OR '.' TO TERMINATE)
```

市町村名を入力して確認した後：

```
RECORDS FOUND:        26
ENTER 'D' TO DISPLAY RECORDS D
```

D を入力して確認した後：

```
        NAME              FIRST-NAME         TELEPHONE
-------------------- -------------------- ---------------

MAIZIERE             ELISABETH            46758304
MARX                 JEAN-MARIE           40738871
REIGNARD             JACQUELINE           48472153
RENAUD               MICHEL               46055008
REMOUE               GERMAINE             36929371
LAVENDA              SALOMON              40155905
BROUSSE              GUY                  37502323
GIORDA               LOUIS                37497316
SIECA                FRANCOIS             40487413
CENSIER              BERNARD              38070268
DUC                  JEAN-PAUL            38065261
CAHN                 RAYMOND              43723961
MAZUY                ROBERT               44286899
FAURIE               HENRI                44341159
VALLY                ALAIN                47326249
BRETON               JEAN-MARIE           48467146
GIGLEUX              JACQUES              40477399
KORAB-BRZOZOWSKI     BOGDAN               45288048
XOLIN                CHRISTIAN            46060015
LEGRIS               ROGER                39341509
VVVV
```

61 EXAMINE

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ADD** | **COMPRESS** | **COMPUTE** | **DIVIDE** | **MOVE** | **MOVE ALL** | **MULTIPLY** | **RESET** | **SEPARATE** | **SUBTRACT**

関連機能グループ：「*算術演算とデータ移動操作*」

構文 1 - EXAMINE

```
EXAMINE [DIRECTION-clause]
        [FULL [VALUE [OF]]] { operand1
                            { SUBSTRING
                              (operand1,operand2,operand3)
        POSITION-clause
        [FOR] [FULL [VALUE [OF]]] [PATTERN] operand4
        [DELIMITERS-option]
        {[DELETE-REPLACE-clause] [GIVING-clause]}
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

構文説明 - 構文 1

EXAMINE ステートメントは、英数字やバイナリフィールドの内容、または配列内にあるフィールドの範囲を調べるため、および次の目的で使用します。

- 検索パターンが見つかった回数を返します。
- 検索パターンが最初に出現したバイト位置を返します。
- フィールドの有効な内容の長さ（末尾の空白を除いたフィールド長）を返します。
- パターンが最初に見つかった配列フィールドのオカレンス番号（インデックス）を返します。
- パターンを別のパターンで置換します。
- パターンを削除します。

オペランド定義テーブル：

オペランド	構文要素					フォーマット												オペランド参照	ダイナミック定義
operand1	C*	S	A			A	U					B						可	不可
operand2	C	S						N	P	I		B*						可	不可
operand3	C	S						N	P	I		B*						可	不可
operand4	C	S	A			A	U					B						可	不可

* GIVING 節を使用する場合、*operand1* に定数を指定できます。DELETE または REPLACE 節を使用する場合、定数の指定はできません。

* *operand2* および *operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

DIRECTION-clause	方向節では、検索方向を決めます。 詳細については、下記の「***DIRECTION 節***」を参照してください。
operand1	*operand1* では、内容を調べるフィールドを指定します。 *operand1* が DYNAMIC 変数である場合は、REPLACE 操作でフィールドの長さを拡張または縮小できます。また、DELETE 操作で、フィールド長を"0"に設定することもできます。 DYNAMIC 変数の現在の長さは、システム変数 *LENGTH を使用して確認できます。 DYNAMIC 変数の全般的な情報については、「*ラージ変数／フィールドとダイナミック変数／フィールド*」セクションを参照してください。
POSITION-clause	POSITION 節は、*operand1*（または *operand1* のサブストリング）内の調査の開始位置と終了位置を指定するために使用できます。 詳細については、下記の「***POSITION 節***」を参照してください。
operand4	*operand4* では、EXAMINE 操作に使用する値を指定します。 *operand4* および *operand6* の詳細については、DELETE REPLACE 節で使用される下記の *operand6* を参照してください。
FULL	オペランドに FULL を指定すると、末尾の空白も含めて値全体が処理されます。 FULL を指定しない場合は、オペランド内の末尾の空白は無視されます。
SUBSTRING	通常、フィールドの内容はフィールドの始めから終わりまでまたは空白以外の最後の文字まで調べられます。 SUBSTRING オプションを使用すると、フィールドの特定の部分のみを調べることができます。 SUBSTRING 節のフィールド名（*operand1*）の後に、まず開始位置（*operand2*）、次に調べるフィールド部分の長さ（*operand3*）を指定します。

	例えば、フィールド #A の 5 番目から 12 番目を調べるには、次のように指定します。 `EXAMINE SUBSTRING(#A,5,8).` **注意:** 1. *operand2* を省略すると、開始位置は "1" と見なされます。 2. *operand3* を省略すると、長さはフィールドの開始位置から終わりまでと見なされます。 3. SUBSTRING を DYNAMIC 変数と一緒に使用すると、フィールドは固定長変数と同様に動作します。つまり、DELETE または REPLACE 操作が実行されたかどうかに関係なく、EXAMINE 操作の結果として長さ（*LENGTH）が変更されることはありません。
PATTERN	「ワイルドカード文字」（調べる対象から除外する位置を示す記号）を含む値でフィールドを調べる場合は、PATTERN オプションを使用します。*operand4* では、無視する位置に次の記号を挿入できます。 ■ ピリオド（.）、疑問符（?）、下線（_）は、調べる対象から除外する位置が単一の位置であることを示します。 ■ アスタリスク（*）、パーセント記号（%）は、調べる対象から除外する位置の数がいくつでもよいことを示します。 例：PATTERN 'NAT*AL' で、"NAT" と "AL" を含む、任意の値のフィールドを調べることができます。この場合、"NAT" と "AL" の間は、何文字でも許可されます。つまり、"NATURAL" や "NATIONAL" も "NATAL" と同じ結果になります。
DELIMITERS-option	このオプションは、デリミタを示す値を検索するために使用します。詳細については、下記の「***DELIMITERS オプション***」を参照してください。
DELETE-REPLACE-clause	この節の DELETE オプションは、*operand1* で見つかった各検索値（*operand4*）を削除するために使用します。一方、REPLACE オプションは、*operand6* で指定した値によって *operand1* で見つかった各検索値（*operand4*）を置換するために使用します。 詳細については、下記の「***DELETE REPLACE 節***」を参照してください。
GIVING-clause	詳細については、下記の「***GIVING 節***」を参照してください。

DIRECTION 節

方向節では、検索方向を決めます。

オペランド定義テーブル：

オペランド	構文要素	フォーマット	オペランド参照	ダイナミック定義
operand8	C S	A1	可	不可

構文要素の説明：

FORWARD	FORWARD を指定すると、フィールドの内容は左から右へ向かって調べられます。
BACKWARD	BACKWARD を指定すると、フィールドの内容は右から左へ向かって調べられます。
operand8	*operand8* を指定すると、検索方向は *operand8* の内容によって決まります。 *operand8* は A1 のフォーマット／長さで定義する必要があります。 *operand8* に "F" が含まれている場合、検索方向は FORWARD になります。*operand8* に "B" が含まれている場合、検索方向は BACKWARD になります。" 他の値はすべて無効であり、*operand8* が定数であればコンパイル時に、*operand8* が変数であれば実行時に排除されます。

Note: DIRECTION 節を指定しない場合、デフォルトの方向は FORWARD になります。

POSITION 節

POSITION 節は、*operand1*（または *operand1* のサブストリング）内の調査の開始位置と終了位置を指定するために使用できます。

オペランド定義テーブル：

オペランド	構文要素					フォーマット										オペランド参照	ダイナミック定義
operand9	C	S				N	P	I								可	不可
operand10	C	S				N	P	I								可	不可

構文要素の説明：

FROM *operand9*	*operand9* では、調査の開始位置を指定します。
ENDING AT / **THROUGH** *operand10*	*operand10* では、調査の終了位置を指定します。

開始位置（*operand9*）と終了位置（*operand10*）は *operand1* または *operand1* のサブストリングに関連しており、いずれも処理されます。

検索は開始位置から始まり、終了位置で終わります。

開始および（または）終了位置が指定されない場合、デフォルト値が適用されます。この値は、検索方向によって決定されます。

方向	デフォルト開始位置	デフォルト終了位置
FORWARD	1（最初の文字）	*operand1* の長さ（最後の文字）
BACKWARD	*operand1* の長さ（最後の文字）	1（最初の文字）

この解決策で、`EXAMINE BACKWARD ...` は、`EXAMINE BACKWARD ... FROM *LENGTH(...) THRU 1` と同じであり、予期したとおりに動作します。

Note: 検索方向が `FORWARD` で開始位置が終了位置を超えている場合、または検索方向が `BACKWARD` で開始位置が終了位置未満の場合、検索は実行されません。

DELIMITERS オプション

```
{ ABSOLUTE                    }
{ [WITH DELIMITERS]           }
{ [WITH DELIMITERS] operand5  }
```

オペランド定義テーブル：

オペランド	構文要素	フォーマット	オペランド参照	ダイナミック定義
operand5	C S	A B	可	不可

構文要素の説明：

ABSOLUTE	これはデフォルトのオプションです。 このオプションを指定すると、値が他のどんな文字に囲まれていても、フィールドに対して指定値の絶対検索が行われます。
WITH DELIMITERS	空白または英数字以外の文字で区切られた値を検索するために使用します。
WITH DELIMITERS *operand5*	*operand5* で指定した文字（列）で区切られた値を検索するために使用します。

DELETE-REPLACE 節

```
[AND] { DELETE [FIRST]
      { REPLACE [FIRST] [WITH] [FULL [VALUE [OF]]] operand6 }
```

オペランド定義テーブル：

オペランド	構文要素	フォーマット	オペランド参照	ダイナミック定義
operand6	C S A	A U B	可	不可

構文要素の説明：

DELETE	*operand1* の内容から、検索値（*operand4*）の最初（またはすべて）のオカレンスを削除するために使用します。
REPLACE	*operand1* にある検索値（*operand4*）の最初（またはすべて）のオカレンスを、*operand6* で指定した値に置換します。
FIRST	キーワード FIRST を指定すると、最初の同じ値のみを削除または置換できます。

Notes:

1. REPLACE 操作の結果が *operand1* に入り切らない場合は、エラーメッセージが返されます。
2. *operand1* がダイナミック変数である場合は、REPLACE 操作でフィールドの長さを拡張または縮小できます。また、DELETE 操作で、フィールド長を "0" に設定することもできます。 ダイナミック変数の現在の長さは、システム変数 *LENGTH を使用して確認できます。 ダイナミック変数の一般的な情報については、「*ダイナミック変数の使用*」を参照してください。

operand6

operand4 および *operand6* を 1、2、または 3 次元配列として定義し、*n:m* 置換を実現するためのテーブルとして使用することもできます。*operand4* はソースで、*operand6* は出力置換テーブルです。*operand6* から *operand4* へのデータ転送が有効（*operand4*:=*operand6*）である場合、*operand4* と *operand6* は有効な参照です。 いずれの置換テーブルも、次元ごとのインデックス値の昇順に処理されます。

ソース置換テーブル（*operand4*）の 1 番目の文字列から始めて、*operand1* を検索文字列で検索します。 一致した場合、検索文字列は出力置換テーブル（*operand6*）の対応する文字列によって置換されます。一致しなかった場合は、一致するまで *operand1* の検索が文字列に対して継続されます。 ソーステーブルの 1 番目の文字列が、*operand1* の置換された文字列の直後に始まる次の検索に使用されます。

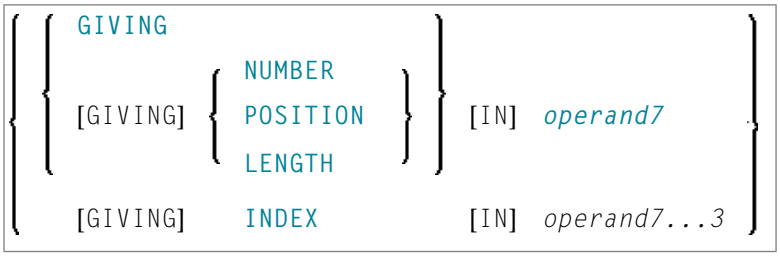
各文字（列）は 1 回だけ文字（列）で置換されます。 つまり、置換された各文字（列）が 2 回置換されることはありません。 置換テーブルをとおしての検索は、一致する文字（列）が見つかるとすぐに終了します。 ランタイムエラーが発生した場合、ソースオペランドは変更されません。

例：

この例では、より小、より大、およびアンパサンド文字に対する HTML 変換を示します。

```
DEFINE DATA LOCAL
1 #HTML  (A/1:3) DYNAMIC INIT <"&lt;", "&gt;", "&amp;">
1 #TAB   (A/1:3) DYNAMIC INIT <"<", ">","&">
1 #DOC(A) DYNAMIC  /* document to be replaced
END-DEFINE
#DOC := "a&lt;&lt;b&amp;b&gt;c&gt;"
WRITE #DOC (AL=30) 'before'
/* Replace #DOC using #HTML to #TAB (n:1 replacement)
EXAMINE   #DOC FOR  #HTML(*)  REPLACE  #TAB(*)
/* "&lt;"  is replaced by "< " (4:1 replacement)
/* "&gt;"  is replaced by ">"  (4:1 replacement)
/* "&amp;" is replaced by "&"  (5:1 replacement)
WRITE #DOC (AL=30) 'after'
END
```

GIVING 節



オペランド定義テーブル：

オペランド	構文要素					フォーマット										オペランド参照	ダイナミック定義
operand7		S					N	P	I							可	可

構文要素の説明：

GIVING	キーワード GIVING だけを指定した場合、これは GIVING NUMBER（デフォルト）に相当します。
NUMBER	内容を調べるフィールド（*operand1*）で検索値（*operand4*）が見つかった回数を取得するために使用します。
POSITION	*operand1*（または *operand1* のサブストリング）内で *operand4* と同じ値が初めて出現したバイト位置を取得するために使用します。
LENGTH	すべての削除または置換操作が完了した後の *operand1*（または *operand1* のサブストリング）に残っている内容の長さを取得するために使用します。 末尾にある空白は無視されます。
operand7	検索値のオカレンス数を指定します。 REPLACE FIRST または DELETE FIRST オプションを同時に使用する場合は、1 を超えないようにする必要があります。
INDEX *operand7...3*	下記を参照してください。

GIVING INDEX

```
[GIVING] INDEX [IN] operand7 ... 3
```

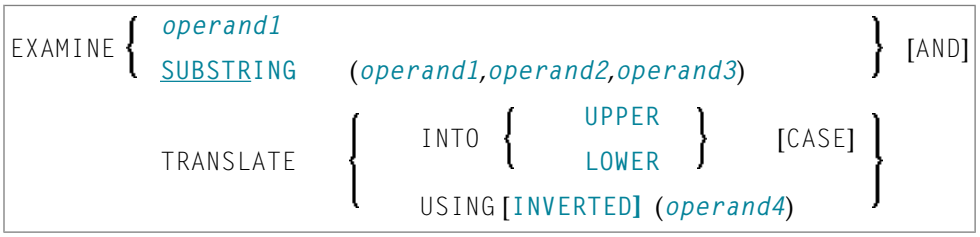
このオプションは、調べられる基礎のフィールドが配列フィールドの場合にのみ適用できます。

構文要素の説明：

INDEX	GIVING INDEX は、最初の検索値（*operand4*）が見つかった *operand1* の配列オカレンス番号（インデックス）を取得するために使用します。
operand7...3	*operand1* の数（最大 3 個）は、 *operand7* の次元数と一致させる必要があります。検索値がどのオカレンスでも見つからなかった場合、 *operand7* には "0" が入ります。

Note: *operand1* のインデックス範囲にオカレンス 0（0：5 など）が含まれている場合、*operand7* の "0" 値は正確ではありません。 この場合、GIVING NUMBER 節を追加して検索値が実際に見つかったかどうかを明確にする必要があります。

構文 2 - EXAMINE TRANSLATE



構文図で使用されている記号については、「*構文記号*」を参照してください。

構文説明 - 構文 2

EXAMINE TRANSLATE ステートメントでは、フィールド内の文字を大文字や小文字、または他の文字に変換します。

オペランド定義テーブル：

オペランド	構文要素			フォーマット	オペランド参照	ダイナミック定義
operand1		S	A	A B	可	不可
operand2	C	S		N P I B*	可	不可
operand3	C	S		N P I B*	可	不可
operand4		S	A	A B	可	不可

**operand2* および *operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

EXAMINE *operand1*	フィールド内容の全体の変換： *operand1* で指定したフィールドの内容が変換されます。
EXAMINE SUBSTRING *operand1* *operand2* *operand3*	フィールド内容の一部分の変換： 通常は、フィールドの内容全体が変換されます。 SUBSTRING オプションを指定すると、フィールドの特定の部分のみが変換されます。 SUBSTRING 節のフィールド名（*operand1*）の後に、まず開始位置（*operand2*）、次に調べるフィールド部分の長さ（*operand3*）を指定します。

	例えば、フィールド #A の 5 番目から 12 番目を変換する場合は、次のように指定します。 `EXAMINE SUBSTRING(#A,5,8) AND TRANSLATE ...` **注意:** *operand2* を省略すると、開始位置は "1" と見なされます。 *operand3* を省略すると、長さはフィールドの開始位置から終わりまでと見なされます。
TRANSLATE INTO UPPER CASE	大文字変換： *operand1* の内容が大文字に変換されます。
TRANSLATE INTO LOWER CASE	小文字変換： *operand1* の内容が小文字に変換されます。
TRANSLATE USING *operand4*	使用する変換テーブル： *operand4* は、文字変換に使用する変換テーブルです。 テーブルのフォーマット／長さは、A2 または B2 にする必要があります。 **注意:** 変換される文字に対して変換テーブルで複数の変換が定義されている場合は、最後の変換が適用されます。
INVERTED	キーワード INVERTED を指定すると、変換テーブル（*operand4*）は逆に使用されます。つまり、変換の方向が逆転します。

構文 3 - Unicode 書記素用の EXAMINE

```
EXAMINE [FULL [VALUE [OF]]] { operand1                                  }
                            { SUBSTRING(operand1,operand2,operand3)     }
[POSITION-clause]
                            {                                           }
[FOR]                       { CHARPOSITION operand4 CHARLENGTH operand5 }
                            { CHARPOSITION operand4                     }
                            { CHARLENGTH operand5                       }
[GIVING] POSITION IN operand6 [[GIVING] LENGTH IN operand7]
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

構文説明 - 構文 3

"grapheme" は、ユーザーが一般に文字と見なすものです。 ほとんどの場合、UTF-16 コード単位（=U形式文字）が書記素になりますが、書記素は複数のコード単位で構成されることもあります。 例えば、一連の基本文字の後に、結合文字またはサロゲートペアが続きます。 書記素および他のUnicode用語の詳細については、***http://www.unicode.org/*** にある『*The Unicode Standard*』を参照してください。

U 形式オペランド用の EXAMINE ステートメントは、通常、コード単位に対して機能します。 ただし、CHARPOSITION および CHARLENGTH 節を使用している場合は、書記素シーケンスの開始位置と長さ（コード単位での）を取得できます。 返されるコード単位の値は、コード単位のオペランドを必要とする他のステートメントまたは節で使用できます（例：SUBSTRING 節）。

EXAMINE ステートメントのこの構文に関する詳細については、「*Natural プログラミング言語での Unicode およびコードページのサポート*」の「*ステートメント*」セクションと「*EXAMINE*」セクションも参照してください。

オペランド定義テーブル：

オペランド	構文要素					フォーマット											オペランド参照	ダイナミック定義
operand1	C	S	A			U				B							可	不可
operand2	C	S					N	P	I	B*							可	不可
operand3	C	S					N	P	I	B*							可	不可
operand4	C	S	A				N	P	I								可	不可
operand5	C	S	A				N	P	I								可	不可
operand6	C	S					N	P	I								可	不可
operand7	C	S					N	P	I								可	不可

* *operand2* および *operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

FULL	オペランドに FULL を指定すると、末尾の空白も含めて値全体が処理されます。 FULL を指定しない場合は、オペランド内の末尾の空白は無視されます。
SUBSTRING *operand1* *operand2* *operand3*	通常、フィールドの内容はフィールドの始めから終わりまでまたは空白以外の最後の文字まで調べられます。 SUBSTRING オプションを使用すると、フィールドの特定の部分のみを調べることができます。 SUBSTRING 節のフィールド名（オペランド1）の後に、まず開始位置（*operand2*）、次に調べるフィールド部分の長さ（*operand3*）を指定します。 *operand2* と *operand3* は、コード単位で指定します。

	例えば、フィールド#Aの5番目から12番目を調べるには、次のように指定します。 `EXAMINE SUBSTRING (#A,5,8)` **注意:** 1. *operand2* を省略すると、開始位置は "1" と見なされます。 2. *operand3* を省略すると、長さはフィールドの開始位置から終わりまでと見なされます。 3. SUBSTRING を DYNAMIC 変数と一緒に使用すると、フィールドは固定長変数と同様に動作します。つまり、DELETE または REPLACE 操作が実行されたかどうかに関係なく、EXAMINE 操作の結果として長さ（*LENGTH）が変更されることはありません。
POSITION-clause	FROM と THRU の位置は、コード単位で指定します。 詳細については、「構文 1」の ***POSITION 節*** を参照してください。
CHARPOSITION *operand4*	*operand4* では、書記素シーケンスの開始位置（Unicode 書記素）を定義します。対応する位置（コード単位）が、*operand6* で返されます。 CHARLENGTH 節が指定されている場合は、この節を省略できます。その場合、開始位置は 1 とみなされます。
CHARLENGTH *operand5*	*operand5* では、書記素シーケンスの長さ（Unicode 書記素）を定義します。 書記素シーケンスの長さ（コード単位）が、*operand7* で返されます。 CHARPOSITION 節が指定されている場合は、この節を省略できます。その場合は、開始位置から文字列の末尾までの長さが返されます。
GIVING POSITION IN *operand6*	*operand6* は、*operand4* と *operand5* によって定義された書記素シーケンスの開始位置（コード単位）を受け取ります。 オペランド 1 が *operand4* 書記素よりも小さい場合は、0 が返されます。 GIVING LENGTH 節が指定されている場合は、この節を省略できます。
GIVING LENGTH IN *operand7*	*operand7* は、*operand4* と *operand5* によって定義された書記素シーケンスの長さ（コード単位）を受け取ります。 *operand1* が *operand4*+*operand5* 書記素よりも小さい場合は、0 が返されます。 GIVING POSITION 節が指定されている場合は、この節を省略できます。

Notes:

1. CHARPOSITION と CHARLENGTH 節の一方または両方を指定する必要があります。
2. GIVING POSITION と GIVING LENGTH 節の一方または両方を指定する必要があります。

例

例 1 - EXAMINE

```
** Example 'EXMEX1': EXAMINE
************************************************************************
DEFINE DATA LOCAL
1 #TEXT   (A40)
1 #A      (A1)
1 #START  (N2)
1 #NMB1   (N2)
1 #NMB2   (N2)
1 #NMB3   (N2)
1 #NMBEX2 (N2)
1 #NMBEX3 (N2)
1 #NMBEX4 (N2)
1 #POSEX5 (N2)
1 #LGHEX6 (N2)
1 #NMBEX7 (N2)
1 #NMBEX8 (N2)
END-DEFINE
*
WRITE 'EXAMPLE 1 (GIVING NUMBER, WITH DELIMITER)'
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
ASSIGN #A = 'A'
EXAMINE #TEXT FOR #A GIVING NUMBER #NMB1
EXAMINE #TEXT FOR #A WITH DELIMITER GIVING NUMBER #NMB2
EXAMINE #TEXT FOR #A WITH DELIMITER '.' GIVING NUMBER #NMB3
WRITE NOTITLE '=' #NMB1 '=' #NMB2 '=' #NMB3
*
WRITE / 'EXAMPLE 2 (WITH DELIMITER, REPLACE, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE #TEXT FOR #A WITH DELIMITER '-' REPLACE WITH '*'
        GIVING NUMBER #NMBEX2
WRITE '=' #TEXT '=' #NMBEX2
*
WRITE / 'EXAMPLE 3 (REPLACE, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE #TEXT ' ' REPLACE WITH '+' GIVING NUMBER #NMBEX3
WRITE '=' #TEXT '=' #NMBEX3
*
WRITE / 'EXAMPLE 4 (FULL, REPLACE, GIVING NUMBER)'
```

```
WRITE '=' #TEXT
EXAMINE FULL #TEXT ' ' REPLACE WITH '+' GIVING NUMBER #NMBEX4
WRITE '=' #TEXT '=' #NMBEX4
*
WRITE / 'EXAMPLE 5 (DELETE, GIVING POSITION)'
WRITE '=' #TEXT
EXAMINE #TEXT '+' DELETE GIVING POSITION #POSEX5
WRITE '=' #TEXT '=' #POSEX5
*
WRITE / 'EXAMPLE 6 (DELETE, GIVING LENGTH)'
WRITE '=' #TEXT
EXAMINE #TEXT FOR 'A' DELETE GIVING LENGTH #LGHEX6
WRITE '=' #TEXT '=' #LGHEX6
*
*
NEWPAGE
*
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
*
ASSIGN #A = 'A B C'
ASSIGN #START = 6
*
WRITE / 'EXAMPLE 7 (SUBSTRING, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE SUBSTRING(#TEXT,#START,9) FOR #A GIVING NUMBER #NMBEX7
WRITE '=' #TEXT '=' #NMBEX7
*
WRITE / 'EXAMPLE 8 (PATTERN, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE #TEXT FOR PATTERN '-A-' GIVING NUMBER #NMBEX8
WRITE '=' #TEXT '=' #NMBEX8
*
END
```

プログラム **EXMEX1** の出力：

```

EXAMPLE 1 (GIVING NUMBER, WITH DELIMITER)
#NMB1:   4 #NMB2:   3 #NMB3:   1

EXAMPLE 2 (WITH DELIMITER, REPLACE, GIVING NUMBER)
#TEXT: ABC   A B C   .A.  .B.  .C.    -A-  -B-
#TEXT: ABC   A B C   .A.  .B.  .C.    -*-  -B-  #NMBEX2:   1

EXAMPLE 3 (REPLACE, GIVING NUMBER)
#TEXT: ABC   A B C   .A.  .B.  .C.    -*-  -B-
#TEXT: ABC+++A+B+C+++.A.++.B.++.C.++++-*-++-B-  #NMBEX3:  18

EXAMPLE 4 (FULL, REPLACE, GIVING NUMBER)
#TEXT: ABC+++A+B+C+++.A.++.B.++.C.++++-*-++-B-
#TEXT: ABC+++A+B+C+++.A.++.B.++.C.++++-*-++-B-+ #NMBEX4:   1
```

```
EXAMPLE 5 (DELETE, GIVING POSITION)
#TEXT: ABC+++A+B+C+++.A.++.B.++.C.++++-*-++-B-+
#TEXT: ABCABC.A..B..C.-*--B-                    #POSEX5:   4

EXAMPLE 6 (DELETE, GIVING LENGTH)
#TEXT: ABCABC.A..B..C.-*--B-
#TEXT: BCBC...B..C.-*--B-                       #LGHEX6:  18
```

例 2 - EXAMINE SUBSTRING、PATTERN、TRANSLATE

```
** Example 'EXMEX2': EXAMINE TRANSLATE
************************************************************************
DEFINE DATA LOCAL
1 #TEXT  (A50)
1 #TAB   (A2/1:10)
1 #START (N2)
END-DEFINE
*
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
*
MOVE 'AX' TO #TAB(1)
MOVE 'BY' TO #TAB(2)
MOVE 'CZ' TO #TAB(3)
*
*
WRITE 'EXAMPLE 1 (USING TRANSLATION TABLE)'
WRITE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING #TAB(*)
WRITE NOTITLE '=' #TEXT
*
WRITE / 'EXAMPLE 2 (USING INVERTED TRANSLATION TABLE)'
WRITE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)
WRITE NOTITLE '=' #TEXT
*
WRITE / 'EXAMPLE 3 (USING SUBSTRING, LOWER CASE)'
WRITE '=' #TEXT
ASSIGN #START = 13
EXAMINE SUBSTRING(#TEXT,#START,15) TRANSLATE INTO LOWER CASE
WRITE '=' #TEXT
END
```

プログラム **EXMEX2** の出力：

```
EXAMPLE 1 (USING TRANSLATION TABLE)
#TEXT: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
#TEXT: XYZ   X Y Z   .X.  .Y.  .Z.    -X-  -Y-  -Z-

EXAMPLE 2 (USING INVERTED TRANSLATION TABLE)
#TEXT: XYZ   X Y Z   .X.  .Y.  .Z.    -X-  -Y-  -Z-
#TEXT: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-

EXAMPLE 3 (USING SUBSTRING, LOWER CASE)
#TEXT: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
#TEXT: ABC   A B C   .a.  .b.  .c.    -A-  -B-  -C-
```

例 3 - EXAMINE TRANSLATE

```
** Example 'EXMEX2': EXAMINE TRANSLATE
************************************************************************
DEFINE DATA LOCAL
1 #TEXT  (A50)
1 #TAB   (A2/1:10)
1 #START (N2)
END-DEFINE
*
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
*
MOVE 'AX' TO #TAB(1)
MOVE 'BY' TO #TAB(2)
MOVE 'CZ' TO #TAB(3)
*
*
WRITE 'EXAMPLE 1 (USING TRANSLATION TABLE)'
WRITE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING #TAB(*)
WRITE NOTITLE '=' #TEXT
*
WRITE / 'EXAMPLE 2 (USING INVERTED TRANSLATION TABLE)'
WRITE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)
WRITE NOTITLE '=' #TEXT
*
WRITE / 'EXAMPLE 3 (USING SUBSTRING, LOWER CASE)'
WRITE '=' #TEXT
ASSIGN #START = 13
EXAMINE SUBSTRING(#TEXT,#START,15) TRANSLATE INTO LOWER CASE
WRITE '=' #TEXT
END
```

プログラム **EXMEX2** の出力：

```
EXAMPLE 1 (USING TRANSLATION TABLE)
#TEXT: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
#TEXT: XYZ   X Y Z   .X.  .Y.  .Z.    -X-  -Y-  -Z-

EXAMPLE 2 (USING INVERTED TRANSLATION TABLE)
#TEXT: XYZ   X Y Z   .X.  .Y.  .Z.    -X-  -Y-  -Z-
#TEXT: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-

EXAMPLE 3 (USING SUBSTRING, LOWER CASE)
#TEXT: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
#TEXT: ABC   A B C   .a.  .b.  .c.    -A-  -B-  -C-
```

例 4 - Unicode 書記素用の EXAMINE

この例では、文字 "ä" と "ü" を含む Unicode 文字列の分析を示します。 この 2 つの文字は基本文字として定義され、その後に結合文字が続きます。"ä" は U+0061 で、その後に U+0308 が続きます。また、"ü" のコードは U+0075 で、その後に U+0308 が続きます。

```
DEFINE DATA LOCAL
1 #U (U20)
1 #START (I2)
1 #POS (I2)
1 #LEN (I2)
END-DEFINE
#U := U'AB'-UH'00610308'-U'CD'-UH'00750308'-U'EF'
*
REPEAT
  #START := #START + 1
  EXAMINE #U FOR CHARPOSITION #START
                   CHARLENGTH      1
               GIVING POSITION IN #POS
                         LENGTH IN #LEN
*
  INPUT (AD=O) MARK POSITION #POS IN FIELD *#U
    '         UNICODE-STRING:' #U     (AD=MI)
 // '          CHARACTER NO.:' #START (EM=9)
  / 'STARTS AT BYTE POSITION:' #POS   (EM=9)
  / '      AND THE LENGTH IS:' #LEN   (EM=9)
WHILE #POS NE 0
```

```
END-REPEAT
END
```

出力：

メインフレーム環境：	Windows、UNIX、および OpenVMS 環境（Web I/O インターフェイスでの出力）：
`UNICODE-STRING: ABa?CDu?EF` `CHARACTER NO.: 1` `STARTS AT BYTE POSITION: 1` `AND THE LENGTH IS: 1`	`UNICODE-STRING: ABäCDüEF` `CHARACTER NO.: 1` `STARTS AT BYTE POSITION: 1` `AND THE LENGTH IS: 1`
Enter キーを押して続行します。	Enter キーを押して続行します。
`UNICODE-STRING: AB?CDu?EF` `CHARACTER NO.: 2` `STARTS AT BYTE POSITION: 2` `AND THE LENGTH IS: 1`	`UNICODE-STRING: ABäCDüEF` `CHARACTER NO.: 2` `STARTS AT BYTE POSITION: 2` `AND THE LENGTH IS: 1`
Enter キーを押して続行します。	Enter キーを押して続行します。
位置 3 にある文字は結合文字シーケンスで、コード単位 2 個分の長さがあることに注意してください。	
`UNICODE-STRING:` `AB<b>a</b>?CDu?EF` `CHARACTER NO.: 3` `STARTS AT BYTE POSITION: 3` `AND THE LENGTH IS: 2`	`UNICODE-STRING: AB<b>ä</b>CDüEF` `CHARACTER NO.: 3` `STARTS AT BYTE POSITION: 3` `AND THE LENGTH IS: 2`
以下、同様。	以下、同様。

62 EXPAND

```
EXPAND { dynamic-clause } [GIVING operand5]
       { array-clause   }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**REDUCE** | **RESIZE**

関連機能グループ：「*ダイナミック変数または X-array のメモリ管理制御*」

機能

EXPAND ステートメントは、拡張のために使用されます。

- ダイナミック変数（*dynamic-clause*）の割り当てられた長さ、または
- X-array（*array-clause*）のオカレンス数

詳細については、『*プログラミングガイド*』の次のセクションを参照してください。

ダイナミック変数の使用

ダイナミック変数のメモリスペースの割り当て／解放

X-array

X-Group 配列のストレージ管理

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													オペランド参照	ダイナミック定義
operand1		S	A			A	U					B							不可	不可
operand2	C	S								I									不可	不可
operand3			A	G		A	U	N	P	I	F	B	D	T	L	C	G	O	可	不可
operand4	C	S						N	P	I									不可	不可
operand5		S								I4									不可	可

構文要素の説明：

dynamic-clause	EXPAND DYNAMIC VARIABLE ステートメントは、ダイナミック変数（*operand1*）の割り当てられた長さを、指定した長さ（*operand2*）に拡張します。 詳細については、後述の「***Dynamic** 節*」を参照してください。
operand1	*operand1* は、サイズを拡張するダイナミック変数です。
operand2	*operand2* では、拡張するダイナミック変数の長さを指定します。 値として、負ではない整数の定数または Integer4（I4）タイプの変数を指定する必要があります。
array-clause	EXPAND ARRAY ステートメントでは、X-array（*operand3*）のオカレンス数を、(*dim*[,*dim*[,*dim*]]) で指定した上下限の範囲内で増加させます。 詳細については、以下の「***Array** 節*」を参照してください。
operand3	*operand3*は、オカレンス数が増やされる X-array です。 配列のインデックス表記はオプションです。 各次元でインデックス表記として使用できるのは、全範囲を示す表記 * のみです。
dim *operand4*	X-array 拡張の上下限表記（*operand4* またはアスタリスク）は、ここで指定します。 現在の上下限の値を使用する必要がある場合は、*operand4*の代わりにアスタリスク（*）を指定できます。 詳細については、後述の「*次元*」を参照してください。
GIVING *operand5*	GIVING 節を指定しない場合は、エラー発生時に Natural ランタイムエラー処理がトリガされます。 GIVING 節を指定した場合は、*operand5* に、エラー発生時は Natural メッセージ番号、成功時はゼロが含まれます。

Dynamic 節

```
[SIZE OF] DYNAMIC [VARIABLE] operand1 TO operand2
```

EXPAND DYNAMIC VARIABLE ステートメントでは、ダイナミック変数（*operand1*）の割り当てられたサイズを、指定した長さ（*operand2*）に拡張します。

*operand2*が、現在割り当てられている *operand1*の長さよりも小さい場合、このダイナミック変数に関してステートメントは無視されます。 現在割り当てられているダイナミック変数のサイズ（*LENGTH）は修正されません。

Array 節

```
[AND RESET [OCCURRENCES OF] ARRAY operand3 TO (dim[,dim[,dim]])
```

EXPAND ARRAY ステートメントは、X-array（*operand3*）のオカレンス数を、TO(*dim*[,*dim*[,*dim*]]) で指定した上下限の範囲内で増加させます。

RESET オプションは、サイズ変更した X-array のすべてのオカレンスをデフォルトのゼロ値にリセットします。 デフォルト（RESET オプションなし）では、実際の値は保持され、サイズ変更した（新しい）オカレンスがリセットされます。

EXPAND ステートメントを使用する場合は、オカレンス数の増加だけが可能です。 要求した数が現在割り当てられているオカレンス数よりも小さい場合は、無視されます。

EXPAND ステートメントで使用される上限または下限は、配列に定義された対応する上限または下限と正確に一致している必要があります。

例：

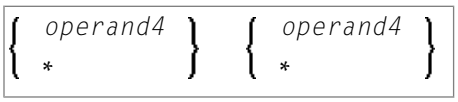
```
DEFINE DATA LOCAL
1 #a(I4/1:*)
1 #g(1:*)
  2 #ga(I4/1:*)

1 #i(i4)
END-DEFINE
...
/* allocating #a(1:10)
EXPAND ARRAY #a TO (1:10)       /* #a is allocated 10
EXPAND ARRAY #a TO (*:10)       /* occurrences.


/* allocating #ga(1:10,1:20)
EXPAND ARRAY #g TO (1:10)       /* 1st dimension is set to (1:10)
EXPAND ARRAY #ga TO (*:*,1:20)  /* 1st dimension is dependent and
                                /* therefore kept with (*:*)
                                /* 2nd dimension is set to (1:20)

EXPAND ARRAY #a TO (5:10)       /* This is rejected because the lower index
                                /* must be 1 or *
EXPAND ARRAY #a TO (#i:10)      /* This is rejected because the lower index
                                /* must be 1 or *

EXPAND ARRAY #ga TO (1:10,1:20) /* (1:10) for the 1st dimension is rejected
                                /* because the dimension is dependent and
                                /* must be specified with (*:*).
```

詳細については、次を参照してください。

- *X-array のストレージ管理*
- *X-Group 配列のストレージ管理*

次元

Array 節で指定する各次元（*dim*）は、次の構文を使用して定義します。



X-array 拡張の上下限表記（*operand4* またはアスタリスク）は、ここで指定します。 現在の上下限の値を使用する必要がある場合は、*operand4* の代わりにアスタリスク（*）を指定できます。 また、"*:*" の代わりに、1 つのアスタリスクを指定することもできます。

次元数（dim）は、X-arrayの定義された次元数（1、2、または3）と正確に一致している必要があります。

指定した次元のオカレンス数が、現在割り当てられているオカレンス数より小さい場合は、対応する次元のオカレンス数は変更されません。

63 FETCH

```
FETCH [ { REPEAT } ] operand1 [operand2 [(parameter)]] ...
      { RETURN }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CALL** | **CALL FILE** | **CALL LOOP** | **CALLNAT** | **DEFINE SUBROUTINE** | **ESCAPE** | **FETCH** | **PERFORM**

関連機能グループ：「*プログラムおよびルーチンの呼び出し*」

機能

FETCH ステートメントは、メインプログラムとして書かれた Natural オブジェクトプログラムを実行します。 ロードするプログラムは、あらかじめ、Natural システムファイルに CATALOG あるいは STOW コマンドで保存しておく必要があります。 FETCH ステートメントの実行によって、Natural ソースワークエリア内にあるソースプログラムが置き換えられることはありません。

Natural RPC については、『*Natural リモートプロシージャコール（RPC）*』ドキュメントの「*サーバーに対する Natural ステートメントの注意事項*」を参照してください。

その他の考慮事項

FETCH 先プログラムは、FETCH で渡されたパラメータの他に、グローバルデータエリアにもアクセスできます。

FETCH ステートメントは、Natural 管理者が設定した Natural プロファイルパラメータ OPRB に基づいて、内部的に END TRANSACTION ステートメントを実行する場合があります。 そのため、1論理トランザクションが複数の Natural プログラムにおよぶ場合は、OPRB パラメータが正しく設定されているかどうかを確認する必要があります。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A												可	不可
operand2	C	S	A	G		A	U	N	P	I	F	B	D	T	L		G	可	可

構文要素の説明：

REPEAT	ユーザー操作の必要性の排除： REPEAT 節は、FETCH されたプログラムの実行中に発行される各 INPUT ステートメントに対して、ユーザーの入力待ちを省略します。 これにより、ユーザーが Enter キーを押すことなく、プログラムの実行についての情報を端末に送ることができます。
RETURN	プログラムタイプのオブジェクトのルーチンとして呼び出しおよび実行： RETURN 指定がないと、FETCH ステートメントを発行したプログラムの実行を終了し、FETCH 先のプログラムが「メインプログラム」（レベル 1）として処理されます。 RETURN を指定すると、FETCH を発行したプログラムの実行は中断され（終了はしない）、FETCH 先のプログラムは「次のレベルのプログラム」として処理が始まります。 そして、FETCH 先のプログラムで END ステートメントや ESCAPE ROUTINE ステートメントが現れると、FETCH RETURN ステートメントを発行したプログラムに制御が戻されます。 処理は FETCH RETURN ステートメントの後のステートメントに渡ります。
operand1	プログラム名： プログラムモジュール名（最大 8 文字）は、英数字定数で指定するか、英数字フォーマットのユーザー定義変数（長さ 1～8）の内容として指定します。 Natural は、FETCH が発行されたときに稼働しているライブラリ内でプログラムを検索します。 プログラムが見つからないとき、Natural は STEPLIB を検索します。 それでも見つからないときには、エラーメッセージを発行します。 プログラム名にアンパサンド（&）を含むこともできます。これは、実行時にシステム変数 *LANGUAGE の現在の値で置き換えられます。 これにより、入力時の言語に応じて入力処理ごとに異なるプログラムを呼び出すことができるようになります。
operand2	パラメータフィールドの引き渡し： FETCH ステートメントで呼び出されるプログラムに、パラメータフィールドを渡すこともできます。 パラメータフィールドはどのフォーマットでも定義できます。 パラメータは、対応する INPUT フィールドに適したフォーマットに変換されます。 すべてのパラメータは、Natural スタックの最上位に配置されます。

	FETCH 先のプログラムは、INPUT ステートメントでパラメータフィールドを読み込みます。FETCH 先のプログラムの最初の INPUT ステートメントで指定したフィールドに、全パラメータフィールド値が入ります。数字フォーマットで定義されたパラメータフィールドに対しては、INPUT ステートメント側で SG=ON の指定が必要です。それは FETCH ステートメントで数字フォーマットで定義された各パラメータフィールドは、値が負の場合は符号桁を持つからです。 次の INPUT ステートメントで読み取られる以上のパラメータが渡されると、余分のパラメータは無視されます。 パラメータの個数は、Natural システム変数 *DATA で取得できます。 **注意:** *operand2* が時刻変数（フォーマット T）の場合は、変数内容のうち時刻コンポーネントのみが渡され、日付コンポーネントは渡されません。
parameter	日付変数の日付形式： *operand2* が日付変数の場合は、この変数に対する *parameter* として、セッションパラメータ DF を指定できます。

例

呼び出し元のプログラム：

```
** Example 'FETEX1': FETCH  (with parameter)
************************************************************************
DEFINE DATA LOCAL
1 #PNUM (N8)
1 #FNC  (A1)
END-DEFINE
*
INPUT 10X 'SELECTION MENU FOR EMPLOYEES SYSTEM' /
      10X '-' (35) //
      10X 'ADD      (A)' /
      10X 'UPDATE   (U)' /
      10X 'DELETE   (D)' /
      10X 'STOP     (.)' //
      10X 'PLEASE ENTER FUNCTION: ' #FNC ///
      10X 'PERSONNEL NUMBER:' #PNUM
*
DECIDE ON EVERY VALUE OF #FNC
  VALUE 'A', 'U', 'D'
    IF #PNUM = 0
      REINPUT 'PLEASE ENTER A VALID NUMBER' MARK *#PNUM
    END-IF
  VALUE 'A'
    FETCH 'FETEXAD' #PNUM
  VALUE 'U'
    FETCH 'FETEXUP' #PNUM
  VALUE 'D'
```

```
    FETCH 'FETEXDE' #PNUM
  VALUE '.'
    STOP
  NONE
    REINPUT 'PLEASE ENTER A VALID FUNCTION' MARK *#FNC
END-DECIDE
*
END
```

呼び出し先のプログラム **FETEXAD**：

```
** Example 'FETEXAD': FETCH (called by FETEX1)
************************************************************************
DEFINE DATA LOCAL
1 #PERS-NR (N8)
END-DEFINE
*
INPUT #PERS-NR
*
WRITE *PROGRAM 'Record added with personnel number:' #PERS-NR
*
END
```

呼び出し先のプログラム **FETEXUP**：

```
** Example 'FETEXUP': FETCH (called by FETEX1)
************************************************************************
DEFINE DATA LOCAL
1 #PERS-NR (N8)
END-DEFINE
*
INPUT #PERS-NR
*
WRITE *PROGRAM 'Record updated with personnel number:' #PERS-NR
*
END
```

呼び出し先のプログラム **FETEXDE**：

```
** Example 'FETEXDE': FETCH (called by FETEX1)
************************************************************************
DEFINE DATA LOCAL
1 #PERS-NR (N8)
END-DEFINE
*
INPUT #PERS-NR
*
WRITE *PROGRAM 'Record deleted with personnel number:' #PERS-NR
```

```
*
END
```

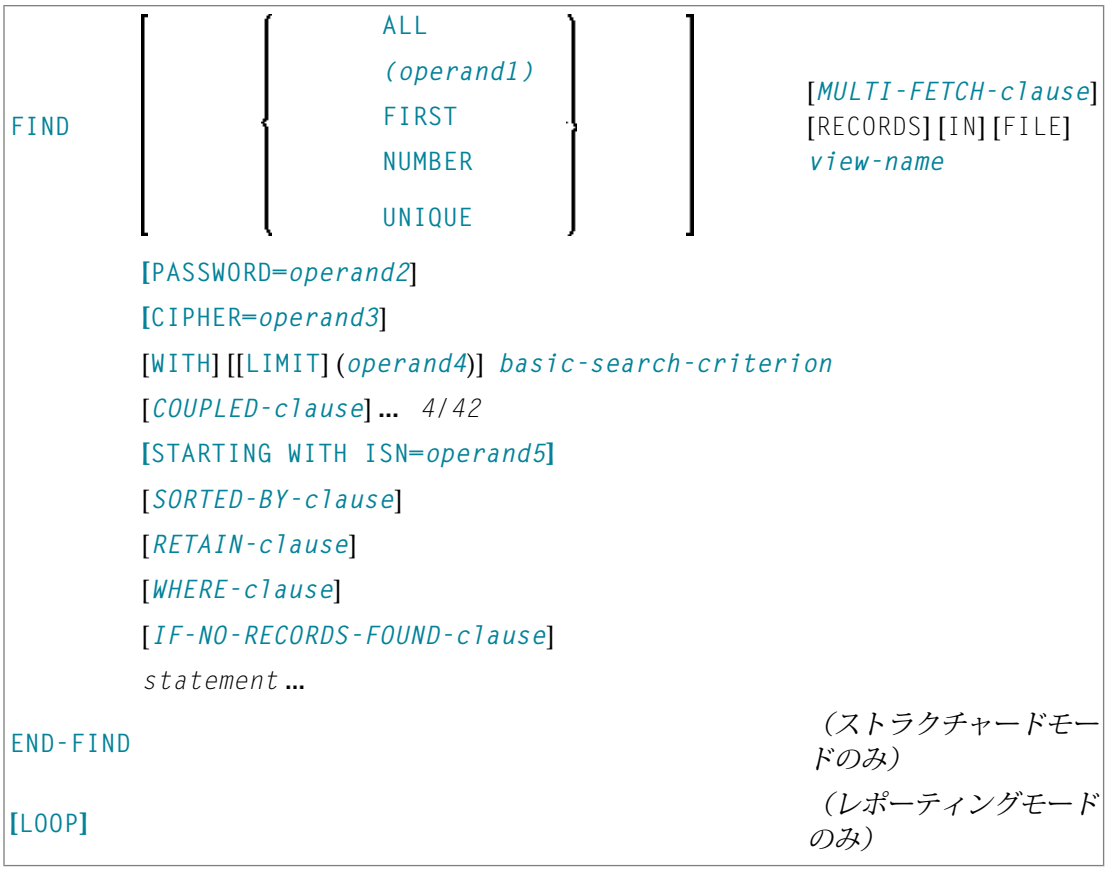
プログラム **FETEX1** の出力：

```

SELECTION MENU FOR EMPLOYEES SYSTEM
-----------------------------------

ADD      (A)
UPDATE   (U)
DELETE   (D)
STOP     (.)

PLEASE ENTER FUNCTION: D


PERSONNEL NUMBER: 1150304
```

機能と社員番号を入力した後：

```

Page      1                                                  05-01-13  11:58:46

FETEXDE  Record deleted with personnel number:   1150304
```

64 FIND



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

FIND ステートメントは、ディスクリプタ（キー）として定義したフィールドの検索条件をもとに、データベースからレコードの集合を選択します。

このステートメントは処理ループを開始させ、選択された各レコードに対してループ内の処理が行われます。 各レコードのすべてのフィールドを処理ループ内で参照できます。 そのため、選択した各レコード内のフィールドを参照するために、FIND ステートメントに続けて READ ステートメントを発行する必要はありません。

『*プログラミングガイド*』の「*FIND ステートメント*」も参照してください。

データベース固有の考慮事項

SQL	FIND FIRST は使用できません。PASSWORD、CIPHER、COUPLED、RETAIN の各節も使用できません。 FIND UNIQUE は許可されていません。 SORTED BY 節は SQL の ORDER BY 節に相当します。 SQL データベーステーブルに対する基本検索条件は Adabas ファイルの場合と同じ方法で指定できます。 本文で使用する用語レコードは SQL 用語の 「行」 に相当します。
XML	FIND FIRST は使用できません。PASSWORD、CIPHER、COUPLED、RETAIN の各節も使用できません。 FIND UNIQUE は許可されていません。 XML データベースに対する基本検索条件は Adabas ファイルの場合と同じ方法で指定できます。本文で使用する用語レコードは XML 用語の 「XML オブジェクト」に相当します。

FIND ステートメントで使用可能な Natural システム変数

発行された FIND ステートメントごとに *ISN、*NUMBER および *COUNTER の各 Natural システム変数が自動的に生成されます。実行中の処理ループの外側や、FIND UNIQUE、FIND FIRST、FIND NUMBER のステートメントで生成されたシステム変数を参照する場合は、参照番号を指定する必要があります。 これらのシステム変数のフォーマットおよび長さは P10 です。これを変更することはできません。

***ISN**	Adabas	現在処理中のレコードの Adabas 内部シーケンス番号（ISN）が *ISN に含まれます。 FIND NUMBER ステートメントでは、*ISN は使用できません。
	Tamino	*ISN には XML オブジェクト ID が含まれます。
	SQL	*ISN は使用できません。
	Entire System Server	*ISN は使用できません。
***NUMBER**	Adabas	システム変数 *NUMBER には、WITH 節で指定された基本検索条件を満たすレコード件数が含まれます。
	Tamino	『*システム変数*』ドキュメントの「*SQL データベースに対する *NUMBER*」を参照してください。
	Entire System Server	*NUMBER は使用できません。
***COUNTER**	システム変数 *COUNTER には、入力した処理ループの回数が含まれます。	

「*例 13 - FIND ステートメントで使用可能な Natural システム変数*」も参照してください。

複数の FIND ステートメント

複数の FIND ステートメントを発行してループのネスト構造を作成することができます。その際に、外側の選択レコードごとに内側のループ処理が行われます。

「*例 14 - 複数の FIND ステートメント*」も参照してください。

制限

Entire System Server では、FIND NUMBER および FIND UNIQUE は使用できません。PASSWORD、CIPHER、COUPLED、RETAIN の各節も使用できません。

構文説明

オペランド定義テーブル：

オペランド	構文要素			フォーマット	ステートメント参照	ダイナミック定義
operand1	C	S		N P I B*	可	不可
operand2	C	S		A	可	不可
operand3	C	S		N	可	不可
operand4	C	S		N P I B*	可	不可
operand5	C	S		N P I B*	可	不可

* *operand1*、*operand4* および *operand5* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

ALL*operand1*	処理制限： 選択された集合から処理するレコード件数は、*operand1* として定数（0～4294967295 の範囲）や数値のユーザー定義変数名をカッコで囲んで指定することで制限できます。 ALL は、オプションで指定してもよく、選択されたレコードをすべて処理することを強調するものです。

	operand1 によって制限が指定された場合、この制限は FIND で開始した処理ループに適用されます。 WHERE 節で除かれたレコードは、この制限値の対象とはなりません。 `FIND (5) IN EMPLOYEES WITH ...` `MOVE 10 TO #CNT(N2)` `FIND (#CNT) EMPLOYEES WITH ...` このステートメントでは、指定した制限が、LIMIT ステートメントで設定した制限より優先されます。 LT パラメータで設定された制限の方が小さい場合は、LT 制限が有効になります。 **注意:** 1. 4 桁のレコード件数を指定する場合は、リーディングゼロを指定してください（0*nnnn*）。それは、カッコで囲まれた 4 桁の数はすべて、ステートメントを参照する行番号であると Natural が解釈するためです。 2. *operand1* は RETAIN 節で保持される ISN 集合のサイズには影響しません。 *operand1* は FIND ループに入るときに評価されます。 *operand1* の値が FIND ループ内で変更されても、この値は処理するレコード数に影響を与えません。
FIND FIRST | FIND NUMBER | FIND UNIQUE	これらのオプションは、 ■ 選択されたレコードの集合の最初の1件を使用する（***FIND FIRST***）、 ■ 選択された集合のレコード件数を得る（***FIND NUMBER***）、 ■ または 1 レコードだけが検索条件を満たすことを保証する（***FIND UNIQUE***）ために使用します。 これらのオプションの詳細な説明については、下記を参照してください。
MULTI-FETCH-clause	Adabas データベースの場合、Natural はデータベースアクセス当たり 1 回以上の読み取りを許可する MULTI-FETCH 節を提供します。 詳細については、「***MULTI-FETCH 節***」を参照してください。
view-name	DEFINE DATA ステートメント内、またはグローバルデータエリアやローカルデータエリアで定義されたビュー名です。 レポーティングモードでは、*view-name* を DDM の名前にすることもできます。
PASSWORD=*operand2*	**PASSWORD 節：** PASSWORD 節は、Adabas データベースにのみ適用されます。 この節は Entire System Server では使用できません。

	PASSWORD 節は、パスワード保護された Adabas ファイルのデータを読み込むときまたは書き込むときに、そのパスワード（*operand2*）を指定するために使用します。 パスワードで保護されたファイルにアクセスするときは、データベースセキュリティ責任者にパスワードの使用や割り当てについて相談してください。 パスワードを定数で指定する場合、PASSWORD 節は必ずソースコード行の第1桁目からコーディングします。また、キーワード"PASSWORD"と等号の間には空白を入れないでください。そうすれば、プログラムのソースコードをリストしたときに、サイファキーはリストに出力されません。 TP モードでは、PASSWORD 節を打ち込む前に %* 端末コマンドを入力すれば、パスワードは表示されません。 PASSWORD節を省略すると、PASSW ステートメントで指定したパスワードが適用されます。 パスワードの値は、処理ループの実行中に変更しないでください。 「*例 1 - PASSWORD 節*」も参照してください。
CIPHER=*operand3*	**CIPHER** 節： CIPHER 節は Adabas データベースにのみ適用されます。 この節は Entire System Server では使用できません。 CIPHER節は、サイファリング（暗号化）された Adabas ファイルからデータを読み込むとき、そのサイファキー（*operand3*）を指定します。 サイファリングされたファイルにアクセスするときは、データベースセキュリティの責任者にサイファキーの使用や割り当てについて相談してください。 サイファキーは、数値定数（8 桁）またはユーザー定義変数（N8）で指定できます。 サイファキーを定数で指定する場合、CIPHER 節は必ずソースコード行の第1桁目からコーディングします。そうすれば、プログラムのソースコードをリストしたときに、サイファキーはリストに出力されません。 TP モードでは、CIPHER 節を打ち込む前に %* 端末コマンドを入力すると、CIPHER 節は表示されません。 サイファキーの値は、FIND ステートメントで開始された処理ループ内では変更しないでください。 「*例 2 - CIPHER 節*」も参照してください。
WITH LIMIT *operand4* *basic-search-criterion*	**WITH** 節： WITH 節は必須です。 データベースで定義されているキーフィールド（ディスクリプタ）から構成される basic-search-criterion を指定するために使用されます（「*Adabas ファイルに対する基本検索条件*」を参照）。

	次のデータベース固有の考慮事項が適用されます。 WITH節でディスクリプタ、サブディスクリプタ、スーパーディスクリプタ、ハイパーディスクリプタ、およびフォネティックディスクリプタを使用できます。非ディスクリプタ（DDMに"N"と設定されたフィールド）も指定できます。 WITH 節で選択するレコード件数は、キーワード LIMIT とともに数値定数やユーザー定義変数をカッコで囲んだ制限値（*operand4*）で指定できます。選択レコード件数が制限値を超過した場合、エラーメッセージが出力されプログラムが終了します。 **注意:** 制限が4桁の数の場合、リーディングゼロを指定する必要があります（0*nnnn*）。これは、カッコで囲まれた4桁の数は、すべてステートメントを参照する行番号であると Natural が解釈するためです。
COUPLED-clause	この節は、Adabas のカップリング機能を使用した検索を行います。「***COUPLED 節***」を参照してください。
STARTING WITH ISN=*operand5*	この節は、処理を継続する次のレコードを簡単に指定するために、中断されたFINDループ内での再位置決めに使用できます。「***STARTING WITH 節***」を参照してください。
SORTED-BY-clause	SORTED BY 節は、選択したレコードを1個～3個のディスクリプタを基準に Adabas ソートします。「***SORTED BY 節***」を参照してください。
RETAIN-clause	この節を使用して、大規模ファイルの検索結果を以降の処理のために保持しておくことができます。「***RETAIN 節***」を参照してください。
WHERE-clause	この節を使用して、追加の選択条件（*logical-condition*）を指定できます。「***WHERE 節***」を参照してください。
IF-NO-RECORDS-FOUND-clause	IF NO RECORDS FOUND 節を使用すると、WITH 節および WHERE 節で指定された選択条件を満たすレコードがない場合でも、FINDステートメントで始まる処理ループに入ります。「***IF NO RECORDS FOUND 節***」を参照してください。
END-FIND	FIND ステートメントを終了するには、Natural の予約キーワード END-FIND を使用する必要があります。

FIND FIRST

FIND FIRST ステートメントは、WITH 条件および WHERE 条件を満たす最初の 1 件のレコードを選択し、処理します。

Adabas データベースでは、選択レコードのうち最も低い Adabas ISN 値を持つレコードが処理されます。

このステートメントは処理ループを開始しません。

制限

- FIND FIRST はレポーティングモードでのみ使用可能です。
- FIND FIRST は、 SQL データベースでは使用できません。
- IF NO RECORDS FOUND 節は FIND FIRST ステートメントでは使用できません。

FIND FIRSTで使用可能なシステム変数

次の Natural システム変数が FIND FIRST ステートメントで使用できます。

***ISN**	システム変数 *ISN には選択レコードの Adabas ISN が含まれます。 WITH および WHERE 条件の評価後にレコードが 1 件も見つからなかった場合、*ISN は "0" になります。 あるいは Entire System Server では、*ISN を使用できません。
***NUMBER**	システム変数 *NUMBER には、WITH 条件の評価後、WHERE 条件が評価される前のレコード数が含まれます。 WITH 条件を満たすレコードがなければ、*NUMBER は "0" です。 Entire System Server では *NUMBER を使用できません。
***COUNTER**	レコードが見つかった場合は "1"、レコードが見つからなかった場合は "0" が、システム変数 *COUNTER に含まれます。

FIND FIRST ステートメントの例：**FNDFIR** プログラム（レポーティングモード）を参照してください。

FIND NUMBER

FIND NUMBER ステートメントは、指定した WITH/WHERE 条件を満たすレコード件数を得るために使用します。 処理ループは開始されず、*データベースのフィールドも使用できません。*

Note: WHERE 節を使用すると、負荷が生じることがあります。

制限

- FIND NUMBER ステートメントに、SORTED BY や IF NO RECORDS FOUND の各節を指定することはできません。
- ストラクチャードモードでは、WHERE 節を使用できません。

- FIND NUMBER は、Entire System Server では使用できません。

FIND NUMBER で使用可能なシステム変数

次の Natural システム変数が FIND NUMBER ステートメントで使用できます。

***NUMBER**	システム変数 *NUMBER には、WITH 条件の評価後に見つかったレコード数が含まれます。
***COUNTER**	システム変数 *COUNTER には、WHERE 条件の評価後に見つかったレコード数が含まれます。 *COUNTER は、FIND NUMBER ステートメントに WHERE 節が指定された場合にのみ有効です。

FIND NUMBER の例：**FNDNUM** プログラム（レポーティングモード）を参照してください。

FIND UNIQUE

FIND UNIQUE ステートメントは、処理のために選択されたレコードが1件だけであることを保証するために使用します。 処理ループは開始しません。 WHERE 節の指定がある場合、WHERE 節を評価する内部処理ループが自動的に生成されます。

条件を満たすレコードがないとき、あるいは条件を満たすレコードが複数あるときは、エラーメッセージが返されます。 ON ERROR ステートメントで、エラー条件をチェックできます。

FIND UNIQUE で使用できるシステム変数

***ISN**	システム変数 *ISN は、レコードの一意な ISN 番号を持ちます。レコード自体が一意である必要があります。
***NUMBER**	システム変数 *NUMBER は、有効な FIND UNIQUE 実行に対しては常に 1 です。 エラーが発生した場合、*NUMBER は、他の値（=0 または >=2）を含んでいる可能性があります。 このエラー条件は、ON ERROR ステートメントで使用することができます。 WHERE 節が指定されていない場合、*NUMBER は評価されません。
***COUNTER**	システム変数 *COUNTER には、WHERE 条件の評価後に見つかったレコード数が含まれます。WHERE 節が指定されていない場合、*COUNTER は評価されません。

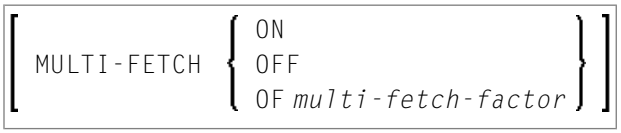
制限事項

- FIND UNIQUE はレポーティングモードでのみ使用可能です。
- FIND UNIQUE は、Entire System Server では使用できません。
- SQL データベースに、FIND UNIQUE は使用できません（例外：メインフレーム環境では、FIND UNIQUE はプライマリキーに対して使用できますが、これは互換性のためだけであるため使用しないでください）。
- SORTED BY や IF NO RECORDS FOUND の各節を FIND UNIQUE ステートメントで使用しないでください。

FIND UNIQUE の例：**FNDUNQ** プログラム（レポーティングモード）を参照してください

MULTI-FETCH 節

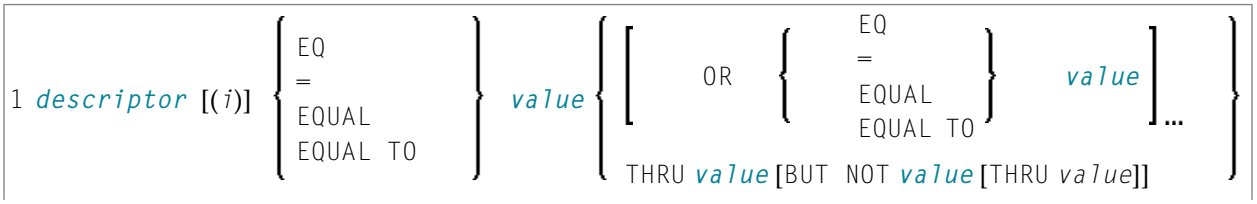
Note: この節は、Adabas データベースでのみ使用できます。

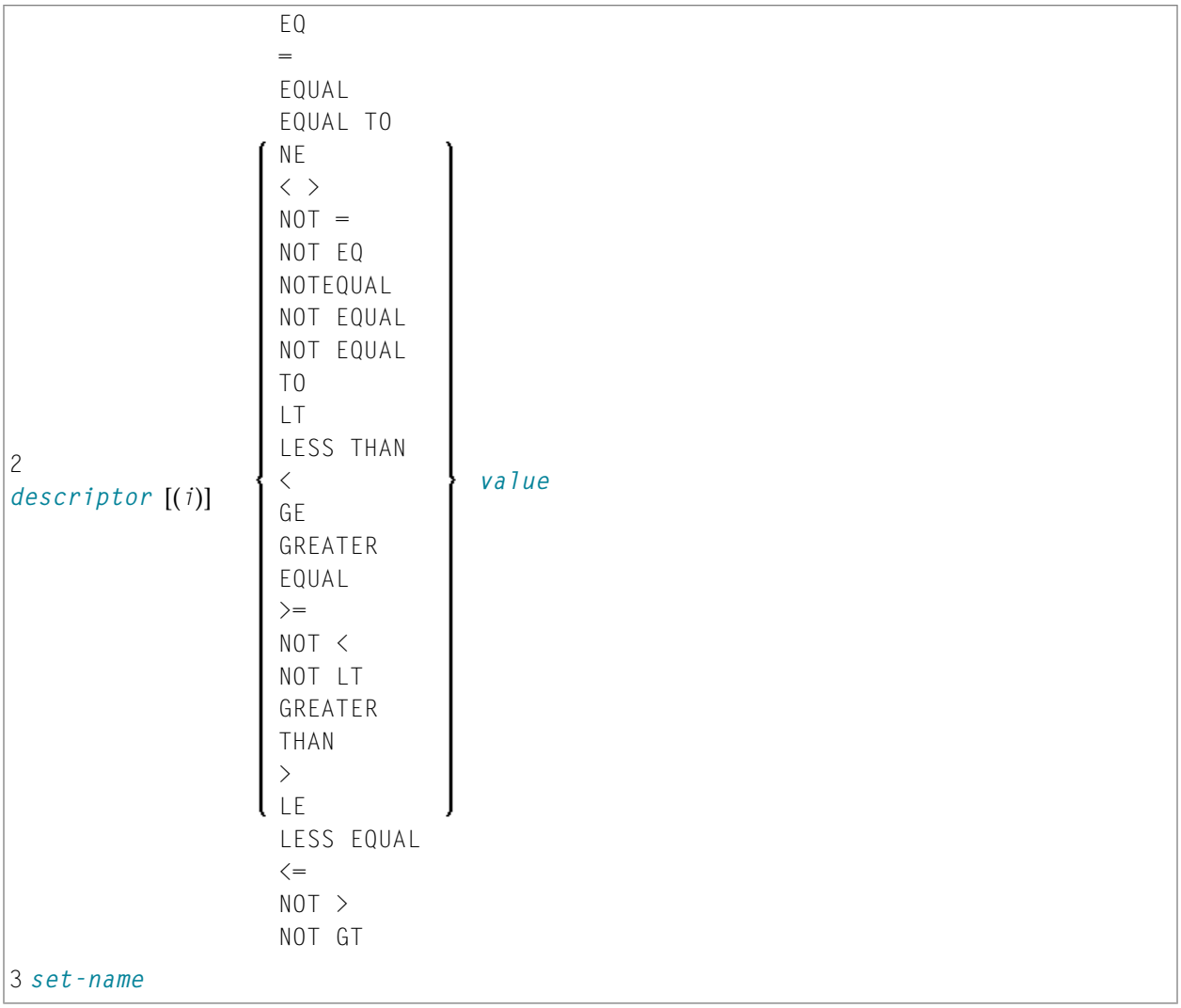


Note: [MULTI-FETCH OF *multi-fetch-factor*] は、データベースタイプ ADA および ADA2 に対しては評価されません。 デフォルト処理モードが適用されます（プロファイルパラメータ MFSET を参照）。 MULTI-FETCH 節は、データベースタイプ ADA2 で使用しても完全に無視されます（『*コンフィグレーションユーティリティ*』ドキュメントの「*データベース管理システムの割り当て*」を参照）。

詳細については、『*プログラミングガイド*』の「*MULTI-FETCH 節*」（Adabas）または。

Adabas ファイルに対する基本検索条件





オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
descriptor		S	A			A	U	N	P	I	F	B	D	T	L				不可	不可
value	C	S				A	U	N	P	I	F	B	D	T	L				可	不可
set-name	C	S				A													不可	不可

構文要素の説明：

descriptor	Adabasディスクリプタ、サブディスクリプタ、スーパーディスクリプタ、ハイパーディスクリプタ、またはフォネティックディスクリプタ。DDMでディスクリプタ以外としてマークされたフィールドも指定できます。
(*i*)	ピリオディックグループに含まれるディスクリプタをインデックス付きまたはなしで指定できます。 インデックスなしで指定した場合、レコードは指定した値がいずれかのオカレンスに見つかった場合に選択されます。 インデックス付きで指定した場合、レコードは値がインデックスで指定したオカレンス内に見つかった場合にのみ選択されます。 インデックスは定数で指定する必要があります。 インデックス範囲は使用できません。 マルチプルバリューフィールドのディスクリプタは、インデックスなしで指定する必要があります。レコードは、値がレコード内で位置に関係なく見つかった場合に選択されます。
value	検索条件値。 ディスクリプタと検索値のフォーマットは変換可能にする必要があります。
set-name	RETAIN節を指定したFINDステートメントで、すでに選択されたレコードの集合の名前です。 FINDで参照する集合名は、同じ物理Adabasファイルから作成する必要があります。 *set-name*は、最大32文字の文字定数、あるいは英数字フォーマットの変数として指定します。 *set-name*は、Entire System Serverでは使用できません。

以下の項目も参照してください。

- *例 3 - WITH 節の基本検索条件*
- *例 4 - マルチプルバリューフィールドの基本検索条件*

NULL インジケータでの検索条件

```
null-indicator { =      } value
               { EQ     }
               { EQUAL [TO] }
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット										ステートメント参照	ダイナミック定義
null-indicator		S							I							不可	不可
value	C	S					N	P	I	F	B					可	不可

構文要素の説明：

null-indicator	空値インジケータ。	
value	指定可能な値	
	-1	対応するフィールドは値を含んでいません。
	0	対応するフィールドは値を含んでいます。

検索条件の結合（**Adabas** ファイル使用時）

Basic-search-criteria は、ブール演算子 AND、OR および NOT を使用して結合できます。カッコで囲んで評価の優先順位を制御することもできます。 評価の順序は次のとおりです。

1. ( )：カッコ
2. NOT：否定（[2] の形式の *basic-search-criterion* にのみ可）
3. AND：AND 結合
4. OR：OR 結合

Basic-search-criteria は論理演算子で複合 *search-expression* の形に結合できます。 複合 *search-expression* の構文は次のようになります。

```
[NOT] { basic-search-criterion } [ { OR  } search-expression ]...
      { (search-expression)     }   { AND }
```

「*例 5 - WITH 節のさまざまな複合検索式の例*」も参照してください。

ディスクリプタ（キー）の使用

Adabas ユーザーは、ディスクリプタとして定義されているデータベースフィールドを基本検索条件に使用できます。

サブディスクリプタ、スーパーディスクリプタ、ハイパーディスクリプタ、フォネティックディスクリプタ

Adabas では、検索条件にサブディスクリプタ、スーパーディスクリプタ、ハイパーディスクリプタ、およびフォネティックディスクリプタが使用できます。

- サブディスクリプタは、フィールドの一部で作られたディスクリプタです。
- スーパーディスクリプタは、1 つ以上のフィールドまたはフィールドの一部から構成されたディスクリプタです。
- ハイパーディスクリプタは、ユーザー定義アルゴリズムを使用して作ったディスクリプタです。

- フォネティックディスクリプタは、発音による検索（例：人の名前）を行うディスクリプタです。 フォネティック検索では、発音が検索値に類似するすべての値が返されます。

どのファイルのどのフィールドをディスクリプタ、サブディスクリプタ、スーパーディスクリプタ、ハイパーディスクリプタ、フォネティックディスクリプタとして使用できるかは対応するDDM に定義されます。

サブディスクリプタ、スーパーディスクリプタ、フォネティックディスクリプタに対する値

これらのディスクリプタ値は、ディスクリプタの内部フォーマットに対して変換できる必要があります。 サブディスクリプタの内部フォーマットは、そのサブディスクリプタを作り出したもとのフィールドのフォーマットと同じです。 スーパーディスクリプタの内部フォーマットは、もとのフィールドがすべて数字フォーマットのときはバイナリであり、そうでなければ英数字フォーマットです。 フォネティックディスクリプタは、常に英数字フォーマットです。

サブディスクリプタやスーパーディスクリプタの値は、次のように指定できます。

- 数字または 16 進定数。 ただしバイナリフォーマットのスーパーディスクリプタに対しては、16 進定数を使用する必要があります（上記参照）。
- サブディスクリプタやスーパーディスクリプタの各部分を、`REDEFINE` ステートメントで再定義した形のユーザー定義変数内の値。

データベース配列内のディスクリプタの使用

データベース内の配列に含まれるディスクリプタも、基本検索条件の中に使用できます。 Adabas データベースを使用している場合、MU または PE に含まれるディスクリプタのことです。

ピリオディックグループに含まれるディスクリプタをインデックス付きまたはなしで指定できます。 インデックスなしで指定した場合、レコードは指定した値がいずれかのオカレンスに見つかった場合に選択されます。 インデックス付きで指定した場合、レコードは値がインデックスで指定したオカレンス内に見つかった場合にのみ選択されます。 インデックスは定数で指定する必要があります。 インデックス範囲は使用できません。

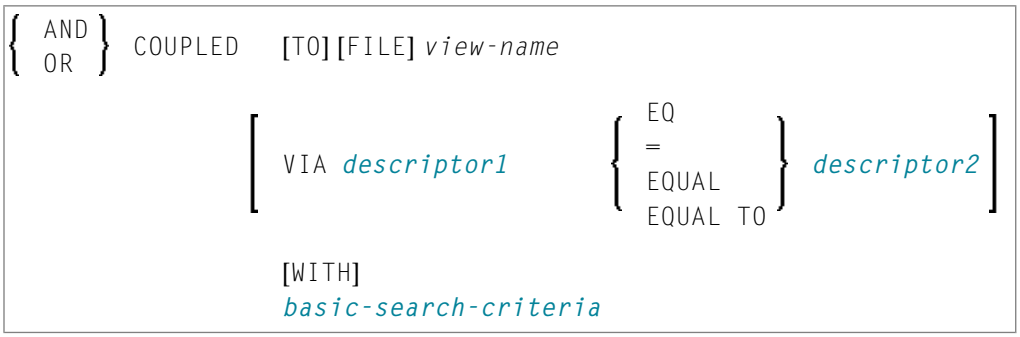
マルチプルバリューフィールドのディスクリプタは、インデックスなしで指定する必要があります。 レコードは、値がレコード内で位置に関係なく見つかった場合に選択されます。

「*例 6 - データベース配列のさまざまな使用例*」も参照してください。

COUPLED 節

この節は Adabas データベースにのみ適用されます。

この節は Entire System Server では使用できません。



オペランド定義テーブル：

オペランド	構文要素					フォーマット									ステートメント参照	ダイナミック定義
descriptor1		S	A			A	N	P			B					不可	不可
descriptor2		S	A			A	N	P			B					不可	不可

Note: VIA 節を指定しない場合は、COUPLED 節を 4 回まで指定できますが、VIA 節を使用すると、42 回まで指定できます。

COUPLED 節は、Adabas のカップリング機能を使用した検索を行います。 この機能により、1 つの FIND ステートメントの検索条件の中に、異なるファイルのデータベースディスクリプタを指定することができます。

同じ Adabas ファイルを、1 つの FIND ステートメントの異なる FIND COUPLED 節に使用しないでください。

set-name（***RETAIN** 節*参照）を *basic-search-criteria* に指定しないでください。

COUPLED 節内で指定したファイルのデータベースフィールドは、別の FIND や READ ステートメントをカップリングされたファイルに発行しない限り、プログラム内で後から参照することはできません。

Note: COUPLED 節を使用する場合は、メインの WITH 節を省略できます。 メインの WITH 節を省略するときは、COUPLED 節でキーワード AND/OR を指定しないでください。

物理カップリング（**VIA**指定なし）

VIA を指定しない COUPLED 節に使用するファイルは、所定の Adabas ユーティリティ（Adabas ドキュメントを参照）を使用して物理的にカップリングされている必要があります。

「*例 7 - 物理カップリングファイルの使用*」も参照してください。

上の例で、DISPLAY ステートメントの NAME は、EMPLOYEES ファイル内にあるので参照できますが、MAKE は COUPLED 節に指定された VEHICLES ファイル内にあるので参照できません。

EMPLOYEES と VEHICLES が物理的にカップリングされている場合にのみ、レコードの検索が行われます。

論理カップリング **- VIA** 節

オプション VIA *descriptor1 = descriptor2* により、検索参照で複数の Adabas ファイルの論理カップリングが可能となります。

- *descriptor1* は最初のビューのフィールドであり、
- *descriptor2* は 2 番目のビューのフィールドです。

2 つのファイルは Adabas で物理的にカップリングする必要はありません。 この COUPLED オプションは、Adabas のソフトカップリング機能（Adabas ドキュメントを参照）を使用します。

「*例 8 - VIA 節*」も参照してください。

STARTING WITH 節

この節は、Adabas データベースにのみ適用されます。

operand5 として、Adabas ISN（内部シーケンス番号）を指定するために、この節を使用すると、レコードの選択のための開始値として使用されます。

この節は、処理を継続する次のレコードを簡単に確定するために、処理が中断された FIND ループ内で再位置決めに使用できます。 これは特に、次のレコードをディスクリプタ値で一意に識別できない場合に役立ちます。また、分散クライアント／サーバーアプリケーションで、レコードをサーバープログラムで読み取ってクライアントプログラムで処理し、レコードを順次処理ではなくバッチ処理する場合にも役立ちます。

Note: 実際に使用される開始値は *operand5* の値ではなく、次に高い値になります。

例：

ライブラリ SYSEXSYN のプログラム FNDSISN を参照してください。

SORTED BY 節

この節は、Adabas、Tamino、または SQL データベースにのみ適用されます。

この節は Entire System Server では使用できません。

```
SORTED [BY] descriptor ... 3 [DESCENDING]
```

SORTED BY 節は、選択したレコードを 1 個～3 個のディスクリプタを基準に Adabas ソートします。ソートの順番を制御するディスクリプタと、レコードの選択に使用するディスクリプタは、異なるものを指定できます。

デフォルトで、レコードは*昇順*にソートされます。降順にソートする場合は、キーワード DESCENDING を指定します。 ソートは、レコードを読み込まず、Adabas インバーテッドリストを使用して行われます。

Note: ソート順を制御するディスクリプタの値の数が多くなると負荷も大きくなります。これは、選択された全レコードがリスト内に位置づけられるまで値のリスト全体をスキャンするからです。 大量のレコードをソートするときには、SORT ステートメントを使用してください。

Adabas のソート制限（Adabas ドキュメントの ADARUN の LS パラメータを参照）は、SORTED BY 節を使用するときに影響します。

PE 内のディスクリプタを SORTED BY 節に使用しないでください。 MU（添字指定なし）は指定できます。

SORTED BY 節にディスクリプタ以外も指定できます。 ただし、メインフレーム環境では、この機能は利用できません。

SORTED BY 節を使用する場合、RETAIN 節を一緒に使用しないでください。

「*例 9 - SORTED BY 節*」も参照してください。

STARTING WITH 節と SORTED BY 節を組み合わせて使用する場合の考慮事項

STARTING WITH 節と SORTED BY 節の両方が同じ FIND ステートメントで使用され、基準データベースが Adabas の場合は、次の点を考慮する必要があります。

Adabas for Mainframes の場合

Adabas for Mainframes では、FIND ステートメントは次の手順で実行されます。

1. 検索条件に一致するすべてのレコードが収集され、ISN 順序で配置されます。
2. レコードは、SORTED BY 節で指定されたディスクリプタの順にソートされます。

3. ISN 値が STARTING WITH 節で指定されているレコードは、「sorted-by-descriptor」 レコードリストに配置されます。
4. 手順 3 で見つかったレコードに続くレコードが、FIND ループで返されます。

Adabas for OpenSystems の場合

Adabas for OpenSystems（UNIX、OpenVMS、Windows）では、同じステートメントは次のように実行されます。

1. 検索条件に一致するすべてのレコードが収集され、ISN 順序で配置されます。
2. ISN 値が STARTING WITH 節で指定されているレコードは、「sorted-by-ISN」 レコードリストに配置されます。
3. 手順 2 で見つかったレコードに続くすべてのレコードは、SORTED BY 節で指定されたディスクリプタ順にソートされ、FIND ループで返されます。

例：

次のプログラムがメインフレーム用の Adabas バージョン 8 および UNIX/OpenVMS/Windows 用の Adabas バージョン 6.1 で実行される場合：

```
DEFINE DATA LOCAL
1 V1 VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
1 #ISN (I4)
END-DEFINE
FORMAT NL=5 SG=OFF PS=43 AL=15
*
PRINT   'FIND' (I)
FIND V1 WITH NAME = 'B' THRU 'BALBIN'
  RETAIN AS 'SET1'
  IF *COUNTER = 4 THEN
    #ISN := *ISN
  END-IF
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. SORTED BY NAME' (I)
FIND V1 WITH 'SET1'
  SORTED BY NAME
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. STARTING WITH ISN = ' (I) #ISN (AD=I)
FIND V1 WITH 'SET1'
  STARTING WITH ISN = #ISN
```

```
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. STARTING WITH ISN = ' (I) #ISN (AD=I)
        ' .. SORTED BY NAME' (I)
FIND V1 WITH 'SET1'
  STARTING WITH ISN = #ISN
  SORTED BY NAME
  DISPLAY *ISN V1
END-FIND
END
```

結果は次のとおりです。

Natural for Mainframes（Adabas バージョン 8）での結果	Natural for OpenSystems（Adabas バージョン 6.1）での結果
ISN NAME FIRST-NAME CITY ----- --------------- --------------- --------------- FIND V1 WITH NAME = 'B' THRU 'BALBIN' 12 BAILLET PATRICK LYS LEZ LANNOY 58 BAGAZJA MARJAN MONTHERME 351 BAECKER JOHANNES FRANKFURT 355 BAECKER KARL SINDELFINGEN 370 BACHMANN HANS MUENCHEN 490 BALBIN ENRIQUE BARCELONA 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY FIND .. SORTED BY NAME 370 BACHMANN HANS MUENCHEN 351 BAECKER JOHANNES FRANKFURT 355 BAECKER KARL SINDELFINGEN	ISN NAME FIRST-NAME CITY ----- --------------- --------------- --------------- FIND V1 WITH NAME = 'B' THRU 'BALBIN' 12 BAILLET PATRICK LYS LEZ LANNOY 58 BAGAZJA MARJAN MONTHERME 351 BAECKER JOHANNES FRANKFURT 355 BAECKER KARL SINDELFINGEN 370 BACHMANN HANS MUENCHEN 490 BALBIN ENRIQUE BARCELONA 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY FIND .. SORTED BY NAME 370 BACHMANN HANS MUENCHEN 351 BAECKER JOHANNES FRANKFURT 355 BAECKER KARL

Natural for Mainframes（Adabas バージョン 8）での結果	Natural for OpenSystems（Adabas バージョン 6.1）での結果
58 BAGAZJA MARJAN MONTHERME 12 BAILLET PATRICK LYS LEZ LANNOY 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY 490 BALBIN ENRIQUE BARCELONA FIND .. STARTING WITH ISN = 355 370 BACHMANN HANS MUENCHEN 490 BALBIN ENRIQUE BARCELONA 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY FIND .. STARTING WITH ISN = 355 .. SORTED BY NAME 58 BAGAZJA MARJAN MONTHERME 12 BAILLET PATRICK LYS LEZ LANNOY 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY 490 BALBIN ENRIQUE BARCELONA	SINDELFINGEN 58 BAGAZJA MARJAN MONTHERME 12 BAILLET PATRICK LYS LEZ LANNOY 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY 490 BALBIN ENRIQUE BARCELONA FIND .. STARTING WITH ISN = 355 370 BACHMANN HANS MUENCHEN 490 BALBIN ENRIQUE BARCELONA 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY FIND .. STARTING WITH ISN = 355 .. SORTED BY NAME 370 BACHMANN HANS MUENCHEN 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY 490 BALBIN ENRIQUE BARCELONA

これらのオプション（SORTED BY または STARTING WITH ISN）の 1 つを指定した FIND ステートメントは、ステートメントが実行されたシステムにかかわらず、同じ順序で同じレコードを返します。 一方、両方の節が一緒に使用されている場合、返される結果は、データベースステートメントの処理に使用されている Adabas プラットフォームによって異なります。

したがって、Naturalプログラムが複数のプラットフォームで使用されることを意図している場合は、同じ FIND ステートメントで SORTED BY 節と STARTING WITH ISN 節を組み合わせて使用することは避ける必要があります。

RETAIN 節

この節は Adabas データベースにのみ適用されます。

この節は Entire System Server では使用できません。

```
RETAIN AS operand6
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット										ステートメント参照	ダイナミック定義
operand6	C	S				A										可	不可

構文要素の説明：

RETAIN AS	RETAIN 節を使用すると、大規模ファイルの検索結果を以降の処理のために保持しておくことができます。 選択されたレコードは、Adabas ワークファイルに "ISN-set" として保持されます。 この集合を後続の FIND ステートメントの基本検索条件で指定して、さらにしぼり込んだり、レコードを処理したりできます。 生成された集合は、特定のファイルに属します。他の FIND ステートメントで集合を使用する場合は、同じファイルを処理する必要があります。 集合は、どの Natural プログラムでも参照できます。
operand6	集合名： 集合名は、レコードの集合を識別するために使用します。英数字定数あるいは英数字のユーザー定義変数の内容として指定できます。 集合名の重複はチェックされません。そのため、同じ集合名を指定すると、新しい集合が古い集合を置き換えます。

「*例 10 - RETAIN 節*」も参照してください。

集合の解放

保持できる集合数や集合内の ISN の件数については、特別な制限はありません。 ただし、一度に定義する ISN の集合数は、最小限にとどめておくことをお勧めします。不要になった集合は、RELEASE SETS ステートメントで解放してください。

RELEASE ステートメントで集合を解放しないと、保持された集合は自動的には解放されず、Natural セッション全体を通して存在します。また、別のライブラリにログオンするまで存在します。 あるプログラムで作成した集合を、他のプログラムで処理したり、検索条件を追加してさらにしぼり込んだりするために参照できます。

他のユーザーによる更新処理

保持された集合内の ISN で識別されたレコードは、他のユーザーのアクセスおよび（または）更新をロックすることはありません。 集合からレコードを処理する前に、集合作成時に使用した検索条件が、その時点でも有効であることを確認するには、FIND ステートメントの基本検索条件として WITH 節に集合名を指定し、WHERE 節に元の検索条件（集合作成時に FIND ステートメントで使用した WITH 節の基本検索条件）を指定します。

制限事項

RETAIN 節を使用する場合、SORTED BY 節を一緒に使用しないでください。

WHERE 節

```
WHERE logical-condition
```

WHERE 節は、追加の選択条件（*logical-condition*）を指定するために使用できます。この条件は、値が読み取られた後、値に対する処理が実行される*前*に評価されます（AT BREAK 評価を含む）。

logical-condition の構文については、『*プログラミングガイド*』の「*論理条件基準*」を参照してください。

WHERE 節を持つ FIND ステートメントに、処理数制限の指定がある場合、WHERE 節により除かれたレコードは制限数の合計数から*除かれます*。しかし、これらのレコードは、Natural セッションパラメータ LT、GLOBALS コマンド、LIMIT ステートメントに指定されたグローバルリミットに対しては、合計されます。

「*例 11 - WHERE 節*」も参照してください。

IF NO RECORDS FOUND 節

ストラクチャードモード構文

```
IF NO    [RECORDS] [FOUND]
       { ENTER       }
       { statement ... }
END-NOREC
```

レポーティングモード構文

```
IF NO [RECORDS] [FOUND]
{    ENTER                }
{    statement            }
{    DO statement ... DOEND }
```

IF NO RECORDS FOUND 節を使用すると、WITH 節および WHERE 節で指定された選択条件を満たすレコードがない場合でも、FIND ステートメントで始まる処理ループに入ります。

IF NO RECORDS FOUND 節は、WITH および WHERE 条件の指定を満たすレコードがない場合、1 回だけ「空」のレコードの FIND 処理ループを実行します。 これが望ましくない場合は、IF NO RECORDS FOUND 節に ESCAPE BOTTOM ステートメントを指定します。

1つ以上のステートメントが IF NO RECORDS FOUND 節で指定されると、処理ループに入る直前にこれらのステートメントが実行されます。 ループに入る前に実行するステートメントがない場合は、キーワード ENTER を使用する必要があります。

「*例 12 - IF NO RECORDS FOUND 節*」も参照してください。

データベース値

IF NO RECORDS FOUND 節内のステートメントで値が割り当てられない限り、Natural はループで指定されたファイルを参照するすべてのデータベースフィールドをリセットして空にします。

システム機能の評価

Natural システム機能は、IF NO RECORDS FOUND 節の結果として処理用に作成される空のレコードに対して 1 回評価されます。

制限事項

この節は、FIND FIRST、FIND NUMBER、および FIND UNIQUE オプションでは使用できません。

例

- 例 12 - IF NO RECORDS FOUND 節
- 例 13 - FIND ステートメントで使用可能な Natural システム変数
- 例 14 - 複数の FIND ステートメント

FIND NUMBER の例（**FNDNUM** プログラム）も参照してください。

例 1 - PASSWORD節

```
** Example 'FNDPWD': FIND (with PASSWORD clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
*
1 #PASSWORD (A8)
END-DEFINE
*
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:' #PASSWORD (AD=N)
LIMIT 2
*
FIND EMPLOY-VIEW PASSWORD = #PASSWORD
                 WITH NAME = 'SMITH'
  DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
*
END
```

プログラム **FNDPWD** の出力：

```
ENTER PASSWORD FOR EMPLOYEE FILE:
```

例 2 - CIPHER 節

```
** Example 'FNDCIP': FIND (with PASSWORD/CIPHER clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
*
1 #PASSWORD (A8)
1 #CIPHER   (N8)
END-DEFINE
*
LIMIT 2
```

```
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:   ' #PASSWORD (AD=N)
    / 'ENTER CIPHER KEY FOR EMPLOYEE FILE: ' #CIPHER   (AD=N)
*
FIND EMPLOY-VIEW PASSWORD = #PASSWORD
                 CIPHER   = #CIPHER
                 WITH NAME = 'SMITH'
  DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
*
END Output of Program FNDCIP:
```

```
ENTER PASSWORD FOR EMPLOYEE FILE:
ENTER CIPHER KEY FOR EMPLOYEE FILE:
```

例 3 - WITH 節の基本検索条件

```
FIND STAFF WITH NAME = 'SMITH'
FIND STAFF WITH CITY NE 'BOSTON'
FIND STAFF WITH BIRTH = 610803
FIND STAFF WITH BIRTH = 610803 THRU 610811
FIND STAFF WITH NAME = 'O HARA' OR = 'JONES' OR = 'JACKSON'
FIND STAFF WITH PERSONNEL-ID = 100082 THRU 100100
                                 BUT NOT 100087 THRU 100095
```

例 4 - マルチプルバリューフィールドの基本検索条件

基本検索条件に使用するディスクリプタが MU のとき、基本的に 4 種類の異なる結果を取得できます。次の例では、MU-FIELD は MU とします。

```
FIND XYZ-VIEW WITH MU-FIELD = 'A'
```

MU-FIELD のいずれかのオカレンスで "A" 値を持っているレコードが選択されます。

```
FIND XYZ-VIEW WITH MU-FIELD NOT EQUAL 'A'
```

MU-FIELD のいずれかのオカレンスで "A" *以外*の値を持っているレコードが選択されます。

```
FIND XYZ-VIEW WITH NOT MU-FIELD NOT EQUAL 'A'
```

MU-FIELD のすべてのオカレンスで "A" 値を持っているレコードが選択されます。

```
FIND XYZ-VIEW WITH NOT MU-FIELD = 'A'
```

MU-FIELD のすべてのオカレンスで "A" *以外*の値を持っているレコードが選択されます。

例 5 - WITH 節のさまざまな複合検索式の例

```
FIND STAFF WITH BIRTH LT 19770101 AND DEPT = 'DEPT06'
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
                    AND (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
                    AND NOT (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH DEPT = 'ABC' THRU 'DEF'
                    AND CITY = 'WASHINGTON' OR = 'LOS ANGELES'
                    AND BIRTH GT 19360101
```

```
FIND CARS WITH MAKE = 'VOLKSWAGEN'
                   AND COLOR = 'RED' OR = 'BLUE' OR = 'BLACK'
```

例 6 - データベース配列のさまざまな使用例

次の例では、SALARY を PE に属するディスクリプタ、LANG を MU であるとします。

```
FIND EMPLOYEES WITH SALARY LT 20000
```

SALARY の全オカレンスを検索します。

```
FIND EMPLOYEES WITH SALARY (1) LT 20000
```

SALARY の第 1 オカレンスだけを検索します。

```
FIND EMPLOYEES WITH SALARY (1:4) LT 20000    /* invalid
```

検索条件に使用されている PE 内のフィールドに範囲を指定しないでください。

```
FIND EMPLOYEES WITH LANG = 'FRENCH'
```

LANG のすべての値を検索します。

```
FIND EMPLOYEES WITH LANG (1) = 'FRENCH'        /* invalid
```

検索条件で使用されている MU に添字を指定しないでください。

例 7 - 物理カップリングファイルの使用

```
** Example 'FNDCPL': FIND (using coupled files)
** NOTE: Adabas files must be physically coupled when using the
**       COUPLED clause without the VIA clause.
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
     AND COUPLED TO
     VEHIC-VIEW WITH MAKE = 'VW'
  DISPLAY NOTITLE NAME
END-FIND
*
END
```

例 8 - VIA 節の使用

```
** Example 'FNDVIA': FIND (with VIA clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
```

```
END-DEFINE
*
FIND EMPLOY-VIEW WITH NAME = 'ADKINSON'
     AND COUPLED TO VEHIC-VIEW
     VIA PERSONNEL-ID = PERSONNEL-ID WITH MAKE = 'VOLVO'
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
END-FIND
*
END
```

プログラム **FNDVIA** の出力：

```
Page      1                                                 05-01-17  13:18:22

PERSONNEL         NAME              FIRST-NAME
   ID
--------- -------------------- --------------------

20011000  ADKINSON             BOB
```

例9 - SORTED BY 節

```
** Example 'FNDSOR': FIND (with SORTED BY clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
END-DEFINE
*
LIMIT 10
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
                 SORTED BY NAME PERSONNEL-ID

  DISPLAY NOTITLE NAME (IS=ON) FIRST-NAME PERSONNEL-ID
END-FIND
*
END
```

プログラム **FNDSOR** の出力：

```
        NAME              FIRST-NAME      PERSONNEL
                                             ID
-------------------- -------------------- ---------

BAECKER              JOHANNES             11500345
BECKER               HERMANN              11100311
BERGMANN             HANS                 11100301
BLAU                 SARAH                11100305
BLOEMER              JOHANNES             11200312
DIEDRICHS            HUBERT               11600301
DOLLINGER            MARGA                11500322
FALTER               CLAUDIA              11300311
                     HEIDE                11400311
FREI                 REINHILD             11500301
```

例 10 - RETAIN 節

```
** Example 'RELEX1': FIND (with RETAIN clause and RELEASE statement)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 BIRTH
  2 NAME
*
1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
     RETAIN AS 'AGESET1'
IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'
  DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
END-FIND
*
RELEASE SET 'AGESET1'
*
END
```

例 **10** の出力：

```
        NAME                 CITY            DATE
                                              OF
                                             BIRTH
-------------------- -------------------- ----------

RUBIN                NEW YORK             1945-10-27
WALLACE              NEW YORK             1945-08-04
```

例 11 - WHERE 節

```
** Example 'FNDWHE': FIND (with WHERE clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 CITY
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'PARIS'
                 WHERE JOB-TITLE = 'INGENIEUR COMMERCIAL'
  DISPLAY NOTITLE
          CITY JOB-TITLE PERSONNEL-ID NAME
END-FIND
*
END
```

プログラム **FNDWHE** の出力：

```
        CITY                  CURRENT          PERSONNEL         NAME
                             POSITION             ID
-------------------- ------------------------- --------- --------------------

PARIS                INGENIEUR COMMERCIAL      50007300  CAHN
PARIS                INGENIEUR COMMERCIAL      50006500  MAZUY
PARIS                INGENIEUR COMMERCIAL      50004700  FAURIE
PARIS                INGENIEUR COMMERCIAL      50004400  VALLY
PARIS                INGENIEUR COMMERCIAL      50002800  BRETON
```

```
PARIS                INGENIEUR COMMERCIAL      50001000  GIGLEUX
PARIS                INGENIEUR COMMERCIAL      50000400  KORAB-BRZOZOWSKI
```

例 12 - IF NO RECORDS FOUND 節

```
** Example 'FNDIFN': FIND (using IF NO RECORDS FOUND)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
EMP. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  /*
  VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMP.)

    IF NO RECORDS FOUND
      MOVE '*** NO CAR ***' TO MAKE
    END-NOREC
    /*
    DISPLAY NOTITLE
            NAME (EMP.) (IS=ON)
            FIRST-NAME (EMP.) (IS=ON)
            MAKE (VEH.)
  END-FIND
  /*
END-READ
END
```

プログラム **FNDIFN** の出力：

```
        NAME              FIRST-NAME              MAKE
-------------------- -------------------- --------------------

JONES                VIRGINIA             CHRYSLER
                     MARSHA               CHRYSLER
                                          CHRYSLER
                     ROBERT               GENERAL MOTORS
                     LILLY                FORD
                                          MG
                     EDWARD               GENERAL MOTORS
                     MARTHA               GENERAL MOTORS
                     LAUREL               GENERAL MOTORS
```

```
                    KEVIN               DATSUN
                    GREGORY             FORD
JOPER               MANFRED             *** NO CAR ***
JOUSSELIN           DANIEL              RENAULT
JUBE                GABRIEL             *** NO CAR ***
JUNG                ERNST               *** NO CAR ***
JUNKIN              JEREMY              *** NO CAR ***
KAISER              REINER              *** NO CAR ***
```

例 13 - FIND ステートメントで使用可能な Natural システム変数

```
** Example 'FNDVAR': FIND (using *ISN, *NUMBER, *COUNTER)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
*
LIMIT 3
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
  DISPLAY NOTITLE PERSONNEL-ID NAME
                  *ISN *NUMBER *COUNTER
END-FIND
*
END
```

プログラム **FNDVAR** の出力

```
PERSONNEL         NAME            ISN         NMBR        CNT
   ID
--------- -------------------- ----------- ----------- -----------

60000114  DE JUAN                      400          41           1
```

```
60000136  DE LA MADRID                 401          41           2
60000209  PINERO                       405          41           3
```

例 14 - 複数の FIND ステートメント

EMPLOYEESファイルをFINDし、姓がSMITHである人をすべて選択します。このEMPLOYEESファイルの PERSONNEL-ID を検索キーにして、VEHICLES ファイルをアクセスします。

```
** Example 'FNDMUL': FIND (with multiple files)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
EMP. FIND EMPLOY-VIEW WITH NAME = 'SMITH'
  /*
  VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = EMP.PERSONNEL-ID
    IF NO RECORDS FOUND
      MOVE '*** NO CAR ***' TO MAKE
    END-NOREC
    DISPLAY NOTITLE
            EMP.NAME (IS=ON)
            EMP.FIRST-NAME (IS=ON)
            VEH.MAKE
  END-FIND
END-FIND
END
```

プログラム **FNDMUL** の出力：

レポートに表示されるのは、姓が SMITH であるすべての人の NAME と FIRST-NAME（姓名）（EMPLOYEESファイルより入手）および彼らの所有する車の MAKE（型）（VEHICLESファイルより入手）です。

```
        NAME             FIRST-NAME             MAKE
-------------------- -------------------- --------------------

SMITH                GERHARD              ROVER
                     SEYMOUR              *** NO CAR ***
                     MATILDA              FORD
                     ANN                  *** NO CAR ***
                     TONI                 TOYOTA
                     MARTIN               *** NO CAR ***
                     THOMAS               FORD
                     SUNNY                *** NO CAR ***
                     MARK                 FORD
                     LOUISE               CHRYSLER
                     MAXWELL              MERCEDES-BENZ
                                          MERCEDES-BENZ
                     ELSA                 CHRYSLER
                     CHARLY               CHRYSLER
                     LEE                  *** NO CAR ***
                     FRANK                FORD
```

65 FOR

```
FOR  operand1        { [:]= }     { operand2                }
                     { EQ   }     { (arithmetic-expression) }
                     { FROM }     { operand2                }
                     { TO   }     { operand3                }
                     { THRU }     { (arithmetic-expression) }
                     [ STEP       { operand4                } ]
                                  { (arithmetic-expression) }
                   statement ...
END-FOR              (structured mode only)
[LOOP]               (reporting mode only)
```

Note: 互換性保持のために、キーワード :=、EQ、FROM、TO、THRU、および STEP は、算術演算ではなく対応する後続のオペランド（*operand2*、*operand3*、または *operand4*）が使用されている場合は省略可能です。

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**REPEAT** | **ESCAPE**

関連機能グループ：「*ループ実行*」

機能

FOR ステートメントは、処理ループを開始し、ループの処理回数を制御するために使用します。

整合性チェック

FOR ループに入る前に、オペランドの値に矛盾がないか（*operand2* に *operand4* を繰り返し加算し、*operand3* の値以上になるか）を確かめます。 値に矛盾があれば、FOR ループには入りません（エラーメッセージは出力されません）。

構文説明

オペランド定義テーブル：

オペランド	構文要素						フォーマット										ステートメント参照	ダイナミック定義
operand1		S					N	P	I	F							可	可
operand2	C	S			N	E	N	P	I	F							可	不可
operand3	C	S			N	E	N	P	I	F							可	不可
operand4	C	S			N	E	N	P	I	F							可	不可

構文要素の説明：

operand1	ループ制御変数（*operand1*）および初期設定（*operand2*） *operand1* は、処理ループの実行回数を制御します。 データベースフィールドまたはユーザー定義変数が使用できます。 キーワード FROM の後の指定値（*operand2*）が、最初に処理ループに入る前に制御変数フィールドに割り当てられます。 制御変数フィールドの値は、ループに入るたびにキーワード STEP の後の指定値（*operand4*）ずつ増えていきます（STEP 値が負数であれば減っていきます）。 処理ループ実行中にループ制御変数を参照できます。ループ制御変数には、その現在値が含まれます。
operand2	
operand3	**TO** 値： 処理ループは、*operand1* が *operand3* の値よりも大きくなると（STEP 値が負であれば小さくなると）終了します。
operand4	**STEP** 値： STEP 値には正でも負でも指定できます。 STEP 値を省略すると、"+1" の増加とみなされます。 ループに入るときに STEP 値の符号によって、「より小さい」 または 「より大きい」 の比較演算が行われます。 *operand4* を "0" にすることはできません。
(*arithmetic-expression*)	*operand2*、*operand3*、または *operand4* の代わりに算術演算を指定できます。 算術式の詳細については、COMPUTE ステートメントの説明にある *arithmetic-expression* を参照してください。 **注意:** 算術演算はカッコで囲む必要があります。
END-FOR	FOR ステートメントを終了するには、Natural の予約語 END-FOR を使用する必要があります。

例

```
** Example 'FOREX1S': FOR (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 #INDEX (I1)
1 #ROOT  (N2.7)
END-DEFINE
*
FOR #INDEX 1 TO 5
  COMPUTE #ROOT = SQRT (#INDEX)
  WRITE NOTITLE '=' #INDEX 3X '=' #ROOT
END-FOR
*
SKIP 1
FOR #INDEX 1 TO 5 STEP 2
  COMPUTE #ROOT = SQRT (#INDEX)
  WRITE '=' #INDEX 3X '=' #ROOT
END-FOR
*
END
```

プログラム **FOREX1S** の出力：

```
#INDEX:    1   #ROOT:   1.0000000
#INDEX:    2   #ROOT:   1.4142135
#INDEX:    3   #ROOT:   1.7320508
#INDEX:    4   #ROOT:   2.0000000
#INDEX:    5   #ROOT:   2.2360679

#INDEX:    1   #ROOT:   1.0000000
#INDEX:    3   #ROOT:   1.7320508
#INDEX:    5   #ROOT:   2.2360679
```

レポーティングモードの例：**FOREX1R**。

66 FORMAT

```
FORMAT [(rep)] parameter ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

FORMAT ステートメントは、入出力時のパラメータ設定を指定します。

FORMAT ステートメントで指定した設定は、GLOBALS コマンドや SET GLOBALS ステートメント（レポーティングモードのみ）、またはNatural管理者が設定した、セッションに有効なデフォルトの設定値を上書きします（コンパイル時）。

DISPLAY、INPUT、PRINT、WRITE、WRITE TITLE、または WRITE TRAILER ステートメントでパラメータを指定すると、この設定値も上書きされます。

設定値は、プログラムの終了または次の FORMAT ステートメントが現れるまで有効です。

FORMAT ステートメントは、Natural 実行コードを生成するものではありません。 実行はプログラムの論理的な処理の流れに依存しません。 このステートメントは、プログラムのコンパイル時に評価されます。つまり、DISPLAY、WRITE、および INPUT ステートメントのコンパイル時のパラメータを設定します。 FORMAT ステートメントで定義された設定は、それに続くすべての DISPLAY、WRITE、および INPUT ステートメントに適用されます。

同一プログラムに、複数の FORMAT ステートメントを使用できますが、レポートごとに 1 つだけ指定します。

構文説明

(*rep*)	レポート指定： 表記（*rep*）は、FORMAT ステートメントを適用するレポートの ID を指定するために使用できます。 範囲 0～31 の値、または **DEFINE PRINTER** ステートメントを使用して割り当てた論理名を指定できます。 （*rep*）を指定しない場合、FORMAT ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。
parameter	パラメータ： パラメータの指定順序は任意で、1 つ以上の空白で区切って指定する必要があります。 1 つのエントリを 2 つのステートメント行に分けて入力しないでください。 ここに適用されるフィールド対応パラメータ設定は、選択されたレポートの INPUT、WRITE、DISPLAY、または PRINT ステートメントで使用される変数フィールドにのみ関連します。これらのステートメントに使用される文字定数には適用しません。 下記の「*アプリケーションパラメータ*」も参照してください。

適用可能なパラメータ

使用できるセッションパラメータの詳細な説明については、『*パラメータリファレンス*』を参照してください。

パラメータ	説明
AD	属性定義
AL	出力の英数字長
CD	カラー定義
DF	日付フォーマット
DL	出力の表示長
EM	編集マスク
ES	空行の省略
FC	充填文字
FL	浮動小数点仮数長

パラメータ	説明
GC	グループに対するヘッダー充填（フィラー）文字
HC	ヘッダーの中央揃え
HW	ヘッダーの幅
IC	挿入文字
IP	入力プロンプトのテキスト
IS	重複抑制
KD	キー定義
LC	先頭文字
LS	行サイズ
MC	マルチプルバリューフィールド数
MP	レポートの最大ページ数
MS	手動による省略
NL	出力の数値長
PC	ピリオディックグループ数
PM	出力モード
PS	ページサイズ
SF	フィールド間の空白
SG	符号の位置
TC	末尾文字
UC	下線付き文字
ZP	ゼロ出力

『*プログラミングガイド*』の「*タイトルおよびヘッダーの下線付き文字 - UC パラメータ*」も参照してください。

例

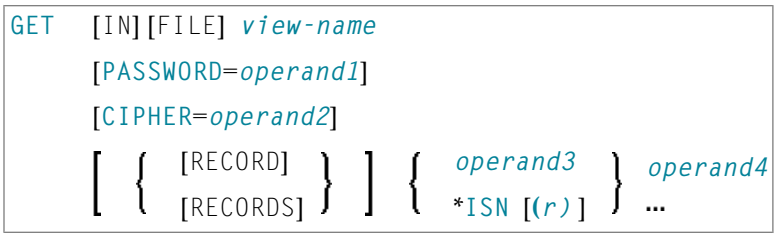
```
** Example 'FMTEX1': FORMAT
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 POST-CODE
  2 COUNTRY
END-DEFINE
*
FORMAT AL=7    /* Alpha-numeric field output length
```

```
       FC=+    /* Filler character for field header
       GC=*    /* Filler character for group header
       HC=L    /* Header left justified
       IC=<<   /* Insert characters
       IS=ON   /* Identical suppress on
       TC=>>   /* Trailing character
       UC==    /* Underline character
       ZP=OFF  /* Zero print off
*
LIMIT 5
READ EMPLOY-VIEW BY NAME
  DISPLAY NOTITLE
          NAME 3X CITY 3X POST-CODE 3X COUNTRY
END-READ
*
END
```

プログラム **FMTEX1** の出力：

```
NAME+++++++   CITY+++++++   POSTAL+++++   COUNTRY++++
                             ADDRESS++++
==========   ==========   ==========   ==========

<<ABELLAN>>   <<MADRID >>   <<28014  >>   <<E  >>
<<ACHIESO>>   <<DERBY  >>   <<DE3 4TR>>   <<UK >>
<<ADAM   >>   <<JOIGNY >>   <<89300  >>   <<F  >>
<<ADKINSO>>   <<BROOKLY>>   <<11201  >>   <<USA>>
              <<BEVERLE>>   <<90211  >>
```

67 GET



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET SAME** | **GET TRANSACTION** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

GET ステートメントは、Adabas ISN（内部シーケンス番号）を指定してレコードを読み込みます。

XML データベースについては、GET ステートメントは、与えられたオブジェクト ID で XML オブジェクトを読み込むために使用されます。

GET ステートメントで処理ループは発生しません。

制限

- SQL データベースに対して GET ステートメントは使用できません。
- Entire System Server では、GET ステートメントを使用できません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A												可	不可
operand2	C	S					N											不可	不可
operand3	C	S			N		N	P	I		B *							可	不可
operand4		S	A			A	N	P	I	F	B	D	T	L				可	可

**operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

view-name	ビュー名： DEFINE DATA ステートメント内またはグローバルデータエリアやローカルデータエリアで定義されたビュー名です。 レポーティングモードでは、*view-name* を DDM の名前にすることもできます。
PASSWORD=*operand1*	Adabas データベースにのみ適用できます。 PASSWORD 節は、パスワード保護されている Adabas ファイルからデータを読み込むときにパスワードを指定します。 CIPHER節は、サイファリング（暗号化）されている Adabas ファイルからデータを読み込むとき、サイファキーを指定するために使用します。 詳細については、ステートメント FIND および PASSW を参照してください。
CIPHER=*operand2*	
***ISN** / *operand3*	内部シーケンス番号： ISN は、数値定数、ユーザー定義変数（*operand3*）の形式、または Natural システム変数 *ISN によって提供されています。
(*r*)	ステートメント参照： 表記（*r*）は、レコードの最初の読み取りに使用される FIND または READ ステートメントを含むステートメントを指定するために使用します。 (*r*) 指定がないと、GET SAME ステートメントは稼働している最も内側の処理ループに関連します。 （*r*）は、参照ステートメント番号またはステートメントラベルとして指定できます。
operand4	データベースフィールドの参照：

	GET ステートメントで読み込んだフィールドを後で参照する場合、GET ステートメントのラベルまたは行番号を指定する必要があります。 *operand4* は、ストラクチャードモードでは使用できません。

例

```
** Example 'GETEX1': GET
************************************************************************
DEFINE DATA LOCAL
1 PERSONS VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 SALARY-INFO VIEW OF EMPLOYEES
  2 NAME
  2 CURR-CODE (1:1)
  2 SALARY    (1:1)
*
1 #ISN-ARRAY  (B4/1:10)
1 #LINE-NR    (N2)
END-DEFINE
*
FORMAT PS=16
LIMIT 10
READ PERSONS BY NAME
  MOVE *COUNTER TO #LINE-NR
  MOVE *ISN     TO #ISN-ARRAY (#LINE-NR)
  DISPLAY #LINE-NR PERSONNEL-ID NAME FIRST-NAME
  /*
  AT END OF PAGE
    INPUT / 'PLEASE SELECT LINE-NR FOR SALARY INFORMATION:' #LINE-NR
    IF #LINE-NR = 1 THRU 10
      GET SALARY-INFO #ISN-ARRAY (#LINE-NR)
      WRITE / SALARY-INFO.NAME
              SALARY-INFO.SALARY    (1)
              SALARY-INFO.CURR-CODE (1)
    END-IF
  END-ENDPAGE
  /*
END-READ
END
```

プログラム **GETEX1** の出力：

```
Page      1                                                 05-01-13  13:17:42

#LINE-NR PERSONNEL         NAME              FIRST-NAME
            ID
-------- --------- -------------------- --------------------

  1      60008339  ABELLAN              KEPA
  2      30000231  ACHIESON             ROBERT
  3      50005800  ADAM                 SIMONE
  4      20008800  ADKINSON             JEFF
  5      20009800  ADKINSON             PHYLLIS
  6      20012700  ADKINSON             HAZEL
  7      20013800  ADKINSON             DAVID
  8      20019600  ADKINSON             CHARLIE
  9      20008600  ADKINSON             MARTHA
 10      20005700  ADKINSON             TIMMIE

PLEASE SELECT LINE-NR FOR SALARY INFORMATION: 1

ABELLAN                1450000 PTA
```

68 GET SAME

ストラクチャードモード構文

```
GET SAME [(r)]
```

レポーティングモード構文

```
GET SAME [(r)] [operand1 ...]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

GET SAME ステートメントは、現在処理中のレコードを再度読み込むために使用します。 このステートメントは、最初にレコードを読む時点で、存在するオカレンスまたは使用するオカレンスの数や範囲がわからない場合に、データベース配列の値（PE または MU）を参照するために最もよく使用します。

制限

- GET SAME ステートメントは Adabas を使用している Natural ユーザーに対してのみ有効です。
- GET SAME ステートメントは Entire System Server では使用できません。
- UPDATE または DELETE ステートメントでは GET SAME ステートメントを参照しないでください。 これらのステートメントは、最初にレコードを読むときに使用した FIND、READ、または GET ステートメントを参照します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1		S	A			A	U	N	P			B						不可	可

構文要素の説明：

(*r***)**	ステートメント参照： 表記（*r*）は、レコードの最初の読み取りに使用される FIND または READ ステートメントを含むステートメントを指定するために使用します。 (*r*) 指定がないと、GET SAME ステートメントは稼働している最も内側の処理ループに関連します。 （*r*）は、参照ステートメント番号またはステートメントラベルとして指定できます。
operand1	参照可能になるフィールド： *operand1* として、GET SAME ステートメントの結果、参照可能になるフィールドを指定します。 **注意:** フィールドを DEFINE DATA ステートメントで定義している場合、*operand1* は指定できません。

例

```
** Example 'GSAEX1': GET SAME
************************************************************************
DEFINE DATA LOCAL
1 I                (P3)
1 POST-ADDRESS VIEW OF EMPLOYEES
  2 FIRST-NAME
  2 NAME
  2 ADDRESS-LINE   (I:I)
  2 C*ADDRESS-LINE
  2 POST-CODE
  2 CITY
*
1 #NAME            (A30)
END-DEFINE
```

```
*
FORMAT PS=20
MOVE 1 TO I
*
READ (10) POST-ADDRESS BY NAME
  COMPRESS NAME FIRST-NAME INTO #NAME WITH DELIMITER ','
  WRITE // 12T #NAME
  WRITE /  12T ADDRESS-LINE (I.1)
  /*
  IF C*ADDRESS-LINE > 1
    FOR I = 2 TO C*ADDRESS-LINE
      GET SAME                      /* READ NEXT OCCURRENCE
      WRITE 12T ADDRESS-LINE (I.1)
    END-FOR
  END-IF
  WRITE / POST-CODE CITY
  SKIP 3
END-READ
END
```

プログラム **GSAEX1** の出力：

```
Page      1                                                  05-01-13  13:23:36



           ABELLAN,KEPA

           CASTELAN 23-C

28014      MADRID







           ACHIESON,ROBERT

           144 ALLESTREE LANE
           DERBY
           DERBYSHIRE

DE3 4TR    DERBY
```

69 GET TRANSACTION DATA

```
GET TRANSACTION [DATA] operand1 ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

`GET TRANSACTION DATA` ステートメントは、以前に `END TRANSACTION` ステートメントで保存したデータを読むために使用します。

`GET TRANSACTION DATA` は、処理ループを開始しません。

システム変数 *ETID

Natural システム変数 `*ETID` は、データベースから読み込むトランザクションデータを識別します。

トランザクションデータが存在しない場合

`GET TRANSACTION DATA` ステートメントが発行されたときに、トランザクションデータが存在しない場合、`GET TRANSACTION DATA` ステートメントで指定した全フィールドは、定義されたフォーマットにかかわりなく全桁が空白で返されます。

Caution: 空のデータに対し算術演算を行うとプログラムは異常終了します。そのため、ユーザーは、「空」のトランザクションデータに対して算術演算を行わないように確認する必要があります。

制限事項

GET TRANSACTION DATA ステートメントは、Adabas データベースにのみ有効です。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1		S				A	U	N	P	I	F	B	D	T				可	可

構文要素の説明：

operand1	フィールド指定： GET TRANSACTION DATA ステートメントに使用するフィールドの順序、長さ、およびフォーマットは、対応する END TRANSACTION ステートメントで指定したフィールドの順序、長さ、およびフォーマットと同じにする必要があります。

例

```
** Example 'GTREX1': GET TRANSACTION
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 MIDDLE-I
  2 CITY
*
1 #PERS-NR (A8) INIT <' '>
END-DEFINE
*
GET TRANSACTION DATA #PERS-NR
IF #PERS-NR NE ' '
  WRITE 'LAST TRANSACTION PROCESSED FROM PREVIOUS SESSION' #PERS-NR
END-IF
```

```
*
REPEAT
  /*
  INPUT  10X 'ENTER PERSONNEL NUMBER TO BE UPDATED:' #PERS-NR
  IF #PERS-NR = ' '
    STOP
  END-IF
  /*
  FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PERS-NR
    IF NO RECORDS FOUND
      REINPUT 'NO RECORD FOUND'
    END-NOREC
    INPUT (AD=M) PERSONNEL-ID (AD=O)
                 / NAME
                 / FIRST-NAME
                 / CITY
    UPDATE
    END TRANSACTION #PERS-NR
  END-FIND
  /*
END-REPEAT
END
```

70 HISTOGRAM

```
HISTOGRAM   [ ALL          ]                    [MULTI-FETCH-clause]
            [ (operand1)   ]                    [multi-fetch-factor] [IN] [FILE] view-name
 [PASSWORD=operand2]
            [                [IN]   { ASCENDING          }                [SEQUENCE] ]
                                    { DESCENDING         }
                                    { VARIABLE operand3  }
                                    { DYNAMIC operand3   }
 [VALUE] [FOR] [FIELD] operand4
 [STARTING/ENDING-clause]
 [WHERE logical-condition]
  statement ...
END-HISTOGRAM（ストラクチャードモードのみ）
[LOOP]（レポーティングモードのみ）
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

HISTOGRAMステートメントは、ディスクリプタ、サブディスクリプタ、またはスーパーディスクリプタとして定義されているデータベースフィールドの値を読むために使用します。値はAdabasインバーテッドリストから直接読み取られます。

HISTOGRAMステートメントは、処理ループを開始します。ただし、HISTOGRAMステートメントで定義したフィールド以外のフィールドを参照することはできません。

『*プログラミングガイド*』の「*HISTOGRAM ステートメント*」も参照してください。

Note: SQL データベースの場合：HISTOGRAM は特定の列に同じ値を持つ行の数を返します。

制限

- XML データベースでは、このステートメントを使用できません。
- Entire System Server では、このステートメントを使用できません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S						N	P	I		B *						可	不可
operand2	C	S				A												可	不可
operand3		S				A												可	不可
operand4		S				A		N	P	I	F	B	D	T	L			不可	不可

**operand1* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

operand1 / **ALL**	処理ループの制限： HISTOGRAM ステートメントで *operand 1* を指定することで処理するディスクリプタ値の数は、数値定数（0～4294967295）またはユーザー定義変数（整数値を含む）を指定して制限できます。 ALL は全ディスクリプタ値が処理されるように強調するためにオプションとして指定できます。 このステートメントでは、指定した制限が、LIMIT ステートメントで設定した制限より優先されます。 LT パラメータで、より小さな制限が設定された場合、LT 制限が適用されます。 **注意:** 4桁の件数を処理する場合、リーディングゼロを指定します（0*nnnn*）。これは、Naturalが、カッコで囲まれた4桁の数をすべてステートメントの参照番号と解釈するためです。*operand1*はHISTOGRAMループに入るときに評価されます。*operand1*の値がHISTOGRAMループ内で変更されても、その値は処理するレコード数に影響を与えません。
MULTI-FETCH-clause	以下の「***MULTI-FETCH 節***」を参照してください。

view-name	*view-name* として、DEFINE DATA ステートメント内、またはグローバルデータエリアやローカルデータエリアで定義したビューの名前です。 ビュー内には HISTOGRAM ステートメントで使用するフィールドだけを指定します（*operand4*）。 ビューのフィールドが、添字範囲付きで定義された PE または MU の場合、その範囲の最初のオカレンスだけが HISTOGRAM ステートメントで参照されます。他のすべてのオカレンスは HISTOGRAM ステートメントの実行には関連しません。 レポーティングモードでは、*view-name* を DDM の名前にすることもできます。
PASSWORD=*operand2*	**PASSWORD** 節： パスワードで保護された Adabas ファイルを検索する場合は、PASSWORD 節にパスワード（*operand2*）を指定します。 詳細については、ステートメント FIND および PASSW を参照してください。
SEQUENCE	**SEQUENCE** 節： この節は、Adabas、SQL データベースに対してだけ使用できます。 この節を使用して、レコードを昇順または降順のどちらで読み取るかを指定できます。 ■ デフォルトは昇順です。昇順はキーワード ASCENDING を使用して明示的に指定することもできます。 ■ レコードを降順で読み取る場合は、キーワード DESCENDING を指定します。 ■ レコードを昇順または降順のどちらで読み取るかを事前に決めずに実行時に決めるには、変数（*operand3*）の前に VARIABLE キーワードまたは DYNAMIC キーワードを指定します。 *operand3* は、フォーマット／長さ A1 で、値 "A"（「昇順」）または "D"（「降順」）が含まれている必要があります。 　■ キーワード VARIABLE が指定された場合、HISTOGRAM 処理ループ開始時に読み込み順（*operand3* の値）が評価されます。*operand3* フィールドが HISTOGRAM ループ内で変更されるかどうかにかかわらず、ループが終了するまで読み込み順は変更されません。 　■ キーワード DYNAMIC が指定された場合、HISTOGRAM 処理ループ内でレコードを取得する前に読み込み順（*operand3* の値）が評価されます。あるレコードからレコードまで読み込み順を変更することが可能です。これは、HISTOGRAM ループの任意の位置で読み込み順を昇順から降順（または、逆）に変更することが可能ということです。 SEQUENCE 節の例： ■ *例 2 - 降順レコード読み取りによる HISTOGRAM ステートメント* ■ *例 3 - 変数順序を使用した HISTOGRAM ステートメント*

operand4	ディスクリプタ： *operand4* として、ディスクリプタ、サブディスクリプタ、スーパーディスクリプタ、またはハイパーディスクリプタが使用できます。 ピリオディックグループに含まれるディスクリプタをインデックス付きまたはなしで指定できます。インデックスなしで指定した場合、ディスクリプタは指定した値がいずれかのオカレンスに見つかった場合に選択されます。インデックス付きで指定した場合、レコードは値がインデックスで指定したオカレンス内に見つかった場合にのみ選択されます。インデックスは定数で指定する必要があります。インデックス範囲は使用できません。 MU のディスクリプタでは、インデックスを指定しないでください。値の位置に関係なく、値がレコードに存在していればディスクリプタは選択されます。	
STARTING-ENDING-clause	**STARTING/ENDING** 節： 開始値と終了値は、キーワード STARTING および ENDING（または、THRU）の後に、処理を開始／終了する値を示す定数またはユーザー定義変数を続けて指定できます。 詳細については、以下の「***Starting/Ending 値の指定***」を参照してください。	
WHERE *logical-condition*	**WHERE** 節： WHERE 節は、追加の選択条件（*logical-condition*）を指定するために使用できます。この条件は、値が読み取られた後、値に対する処理が実行される*前*に評価されます（AT BREAK 評価を含む）。 WHERE 節で指定するディスクリプタは、HISTOGRAM ステートメントで参照したディスクリプタと同じにする必要があります。HISTOGRAM ステートメントでは、他のフィールドに対する処理を行うことができません。 *logical-condition* の構文については、『*プログラミングガイド*』の「*論理条件基準*」で説明しています。 システム変数 Natural システム変数 *ISN、*NUMBER、および *COUNTER が HISTOGRAM ステートメントで使用可能です。 *NUMBER および *ISN は、WHERE 節の評価後に設定されます。これらを WHERE 節の論理条件に使用しないでください。	
	*NUMBER	システム変数 *NUMBER には、最後に読まれた値のデータベースレコード件数が入ります。 SQL データベースについては、『システム変数』ドキュメントの「*SQL データベースに対する *NUMBER*」を参照してください。

	*ISN	システム変数 *ISN には、最後に読まれたディスクリプタ値を持つ、オカレンス番号が入ります。 *ISN は、PE に属さないディスクリプタに対しては常に "0" を返します。 *ISN は、SQL データベースでは使用できません。
	*COUNTER	システム変数 *COUNTER には、読まれた値の個数の合計が入ります（WHERE 節の評価後）。
END-HISTOGRAM	HISTOGRAM ステートメントを終了するには、Natural の予約キーワード END-HISTOGRAM を使用する必要があります。	

MULTI-FETCH 節

Note: この節は、Adabas データベースでのみ使用できます。

```
[ MULTI-FETCH { ON                    } ]
              { OFF                   }
              { OF multi-fetch-factor }
```

Note: [MULTI-FETCH OF *multi-fetch-factor*] は、データベースタイプ ADA および ADA2 に対しては評価されません。 デフォルト処理モードが適用されます（プロファイルパラメータ MFSET を参照）。 MULTI-FETCH 節は、データベースタイプ ADA2 で使用しても完全に無視されます（『*コンフィグレーションユーティリティ*』ドキュメントの「*データベース管理システムの割り当て*」を参照）。

詳細については、『*プログラミングガイド*』の「*MULTI-FETCH 節*」（Adabas）または。

Starting/Ending 値の指定

開始値と終了値は、キーワード STARTING および ENDING（または、THRU）の後に、処理を開始／終了する値を示す定数またはユーザー定義変数を続けて指定できます。

指定された開始値が存在しない場合は、次に高い値が開始値として使用されます。 高い値がまったくないときは、HISTOGRAM ループ内に入りません。

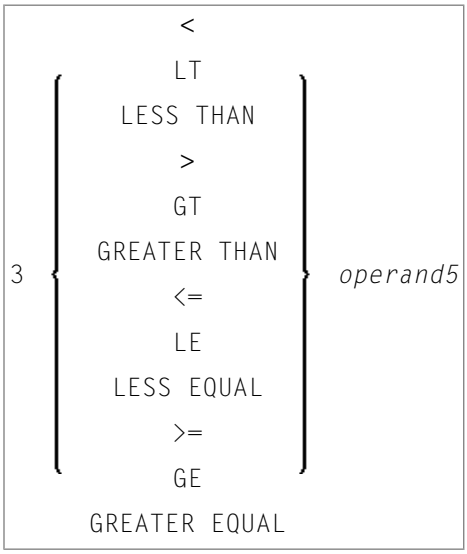
終了値を指定すると、終了値までの（終了値を含む）値が読まれます。

フォーマットが A または B のディスクリプタに対しては、開始値および終了値として 16 進定数を指定できます。

構文オプション 1：

構文オプション 2：

構文オプション 3：



Note: 図 3 の比較演算子を使用した場合は、オプション ENDING AT、THRU、および TO は使用できません。 これらの比較演算子は READ ステートメントにも有効です。

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand5	C	S				A	U	N	P	I	F	B	D	T	L				可	不可
operand6	C	S				A	U	N	P	I	F	B	D	T	L				可	不可

構文要素の説明：

STARTING FROM ... ENDING AT/TO	STARTING FROM および ENDING AT 節は、ユーザー指定の値の範囲に読み込みを制限するために使用します。 STARTING FROM節（=、EQ、EQUAL TO、または[STARTING] FROM）は、読み取り操作の開始値を決めます。 開始値を指定すると、指定した値から読み取りが始まります。 開始値が存在しない場合は、次に高い（またはDESCENDING読み込みでは低い）値が返されます。 より高い（またはDESCENDINGではより低い）値が存在しなければ、HISTOGRAMループに入りません。 値を終了値に制限するには、包括的な範囲を示す用語 THRU、ENDING AT、または TO を持つ ENDING AT 節を指定することができます。 ディスクリプタフィールドが終了値を超えると、自動的なループ終了が実行されます。 TO、THRU および ENDING AT キーワードの基本機能は、非常によく似ていますが、内部的に動作が異なります。
THRU/ENDING AT	THRU または ENDING AT が使用される場合、開始値だけがデータベースに供給され、データベースによって値が返された後に、Naturalランタイムシステムによって終了値のチェックが実行されます。 THRU および ENDING AT 節は、HISTOGRAM ステートメントをサポートするすべてのデータベースに対して使用することができます。
TO	キーワード TO を使用すると、開始値と終了値の両方がデータベースに送られ、Natural は値の範囲をチェックしません。 終了値を超えると、データベースは「ファイルの終わり」に到達した場合と同様に対処し、データベースループを終了します。 データベースにより完全な範囲チェックが行われるので、ASCENDINGまたはDESCENDINGのいずれの順序でブラウズしていても、常に範囲の小さい方の値が開始値になり、大きい方の値が終了値になります。

例

- 例 1 - HISTOGRAM ステートメント
- 例 2 - 降順レコード読み取りによる HISTOGRAM ステートメント
- 例 3 - 変数順序を使用した HISTOGRAM ステートメント

例 1 - HISTOGRAM ステートメント

```
** Example 'HSTEX1S': HISTOGRAM (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
END-DEFINE
*
LIMIT 8
HISTOGRAM EMPLOY-VIEW CITY STARTING FROM 'M'
```

```
  DISPLAY NOTITLE
          CITY 'NUMBER OF/PERSONS' *NUMBER *COUNTER
END-HISTOGRAM
*
END
```

プログラム **HSTEX1S** の出力：

```
        CITY          NUMBER OF     CNT
                       PERSONS
-------------------- ----------- -----------

MADISON                        3           1
MADRID                        41           2
MAILLY LE CAMP                 1           3
MAMERS                         1           4
MANSFIELD                      4           5
MARSEILLE                      2           6
MATLOCK                        1           7
MELBOURNE                      2           8
```

レポーティングモードの例：**HSTEX1R**。

例 2 - 降順レコード読み取りによる HISTOGRAM ステートメント

```
** Example 'HSTDSCND': HISTOGRAM (with DESCENDING)
************************************************************************
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
HISTOGRAM (10) EMPL IN DESCENDING SEQUENCE FOR NAME FROM 'ZZZ'
  DISPLAY NAME *NUMBER
END-HISTOGRAM
END
```

プログラム **HSTDSCND** の出力：

```
Page      1                                                  05-01-13  13:41:03

        NAME            NMBR
-------------------- -----------

ZINN                           1
YOT                            1
YNCLAN                         1
```

```
YATES                          1
YALCIN                         1
YACKX-COLTEAU                  1
XOLIN                          1
WYLLIS                         2
WULFRING                       1
WRIGHT                         1
```

例 3 - 変数順序を使用した HISTOGRAM ステートメント

```
** Example 'HSTVSEQ': HISTOGRAM (with VARIABLE SEQUENCE)
************************************************************************
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
*
1 #DIR      (A1)
1 #STARTVAL (A20)
END-DEFINE
*
SET KEY PF3 PF7 PF8
*
MOVE 'ADKINSON' TO #STARTVAL
*
HISTOGRAM (9) EMPL FOR NAME FROM #STARTVAL
  WRITE NAME *NUMBER
  IF *COUNTER = 5
    MOVE NAME TO #STARTVAL
  END-IF
END-HISTOGRAM
*
#DIR := 'A'
*
REPEAT
  HISTOGRAM EMPL IN VARIABLE #DIR SEQUENCE
            FOR NAME FROM #STARTVAL
    MOVE NAME TO #STARTVAL
    INPUT NO ERASE  (IP=OFF AD=O)
          15/01 NAME *NUMBER
          //   'Direction:' #DIR
          //   'Press PF3 to stop'
          /    '      PF7 to go step back'
          /    '      PF8 to go step forward'
          /    '      ENTER to continue in that direction'
    /*
    IF *PF-KEY = 'PF7' AND #DIR = 'A'
      MOVE 'D' TO #DIR
      ESCAPE BOTTOM
    END-IF
    IF *PF-KEY = 'PF8' AND #DIR = 'D'
```

```
      MOVE 'A' TO #DIR
      ESCAPE BOTTOM
    END-IF
    IF *PF-KEY = 'PF3'
      STOP
    END-IF
  END-HISTOGRAM
  /*
  IF *COUNTER(0250) = 0
    STOP
  END-IF
END-REPEAT
END
```

プログラム **HSTVSEQ** の出力：

```
Page      1                                                 05-01-13  13:50:31

ADKINSON                      8
AECKERLE                      1
AFANASSIEV                    2
AHL                           1
AKROYD                        1
ALEMAN                        1
ALESTIA                       1
ALEXANDER                     5
ALLEGRE                       1









MORE
```

Enter キーを押した後：

```
Page      1                                                  05-01-13  13:50:31

ADKINSON                        8
AECKERLE                        1
AFANASSIEV                      2
AHL                             1
AKROYD                          1
ALEMAN                          1
ALESTIA                         1
ALEXANDER                       5
ALLEGRE                         1



AKROYD                          1

Direction: A

Press PF3 to stop
      PF7 to go step back
      PF8 to go step forward
      ENTER to continue in that direction
```

71 IF

ストラクチャードモード構文

```
IF logical-condition
 [THEN] statement
 [ELSE statement]
END-IF
```

レポーティングモード構文

```
IF logical-condition
 [THEN]   { statement                }
          { DO statement ... DOEND   }
[   ELSE  { statement                } ]
          { DO statement ... DOEND   }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DECIDE FOR** | **DECIDE ON** | **IF SELECTION** | **ON ERROR**

関連機能グループ：「*論理条件の処理*」

機能

IF ステートメントは、論理条件に基づいて、ステートメントあるいはステートメント群の実行を制御します。

Note: 条件に合うときに何もしない場合は、THEN 節に IGNORE ステートメントを指定する必要があります。

構文説明

IF *logical-condition*	IF ステートメントに続いて、指定されたステートメント（群）が実行されるかどうかを判定する論理的な条件です。 例： `IF #A = #B` `IF LEAVE-TAKEN GT 30` `IF #SALARY(1) * 1.15 GT 5000` `IF SALARY (4) = 5000 THRU 6000` `IF DEPT = 'A10' OR = 'A20' OR = 'A30'` 詳細については、『*プログラミングガイド*』の「*論理条件基準*」を参照してください。
THEN *statement*	THEN 節では、論理条件が真である場合に実行される *statement*（群）を指定します。
ELSE *statement*	ELSE 節では、論理条件が真でない場合に実行される *statement*（群）を指定します。
END-IF	IF ステートメントを終了するには、Natural の予約語 END-IF を使用する必要があります。

例

```
** Example 'IFEX1S': IF (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY (1)
  2 BIRTH
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
*
1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19450101' TO #BIRTH (EM=YYYYMMDD)
SUSPEND IDENTICAL SUPPRESS
```

```
LIMIT 20
*
FND. FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
          SORTED BY NAME BIRTH
  IF SALARY (1) LT 40000
    WRITE NOTITLE '*****' NAME 30X 'SALARY LT 40000'
  ELSE
    IF BIRTH GT #BIRTH
      FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (FND.)
        DISPLAY (IS=ON)
                NAME BIRTH (EM=YYYY-MM-DD)
                SALARY (1) MAKE (AL=8)
      END-FIND
    END-IF
  END-IF
END-FIND
END
```

プログラム **IFEX1S** の出力：

```
        NAME            DATE      ANNUAL     MAKE
                         OF       SALARY
                       BIRTH
-------------------- ---------- ---------- --------

BAECKER              1956-01-05      74400 BMW
***** BECKER                                          SALARY LT 40000
BLOEMER              1979-11-07      45200 FIAT
FALTER               1954-05-23      70800 FORD
***** FALTER                                          SALARY LT 40000
***** GROTHE                                          SALARY LT 40000
***** HEILBROCK                                       SALARY LT 40000
***** HESCHMANN                                       SALARY LT 40000
HUCH                 1952-09-12      67200 MERCEDES
***** KICKSTEIN                                       SALARY LT 40000
***** KLEENE                                          SALARY LT 40000
***** KRAMER                                          SALARY LT 40000
```

レポーティングモードの例：**IFEX1R**。

72 IF SELECTION

ストラクチャードモード構文

```
IF SELECTION [NOT UNIQUE [IN [FIELDS]]] operand1 ...
 [THEN] statement ...
 [THEN statement...]
END-IF
```

レポーティングモード構文

```
IF SELECTION [NOT UNIQUE [IN [FIELDS]]] operand1...
        [THEN]  { statement          }
                { DO statement... DOEND }
      [ ELSE    { statement          }       ]
                { DO statement... DOEND }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DECIDE FOR** | **DECIDE ON** | **IF** | **ON ERROR**

関連機能グループ：「*論理条件の処理*」

機能

IF SELECTION ステートメントは、連続して指定された英数字フィールドのうち 1 つのフィールドにのみ値が含まれていることを確認するために使用します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット										ステートメント参照	ダイナミック定義
operand1		S	A			A	U							L	C			可	不可

構文要素の説明：

operand1	選択フィールド： *operand1* として、チェックされるフィールドを指定します 属性制御変数（フォーマットC）を指定する場合は、そのステータスが "MODIFIED" に変更されたかどうかが考慮されます。 属性制御変数にステータス "MODIFIED" が割り当てられているかどうかを確認するには、MODIFIED オプションを使用します。
THEN *statement*	実行するステートメント： キーワード THEN に続くステートメントは、次のどれかが満たされるときに実行されます。 ■ 指定された全フィールド（*operand1*）の値が空値。 ■ 指定フィールドのうち複数フィールド（*operand1*）が空値ではない。 一般的に、このステートメントは、INPUT ステートメントで表示されたマップに対して、端末ユーザーが 1 つの値だけを入力したことを確かめるために使用されます。 **注意:** 条件に合うときに*何もしない*場合は、THEN 節に IGNORE ステートメントを指定します。
END-IF	Natural 予約語 END-IF を使用して、IF SELECTION ステートメントを終了させる必要があります。

例

```
** Example 'IFSEL': IF SELECTION
************************************************************************
DEFINE DATA LOCAL
1 #A (A1)
1 #B (A1)
END-DEFINE
*
INPUT 'Select one function:' //
   9X 'Function A:' #A
   9X 'Function B:' #B
*
IF SELECTION NOT UNIQUE #A #B
  REINPUT 'Please enter one function only.'
END-IF
*
IF #A NE ' '
  WRITE 'Function A selected.'
END-IF
IF #B NE ' '
  WRITE 'Function B selected.'
```

```
1 #A (A1)
1 #B (A1)
END-DEFINE
*
INPUT 'Select one function:' //
   9X 'Function A:' #A
   9X 'Function B:' #B
*
IF SELECTION NOT UNIQUE #A #B
  REINPUT 'Please enter one function only.'
END-IF
*
IF #A NE ' '
  WRITE 'Function A selected.'
END-IF
IF #B NE ' '
  WRITE 'Function B selected.'
END-IF
*
END
```

プログラム **IFSEL** の出力：

```
Select one function:

         Function A:           Function B:
```

機能 **A** を選択して確認した後：

```
Page      1                                                  05-01-17  11:04:07

Function A selected.
```

73 IGNORE

IGNORE

このchapterでは、次のトピックについて説明します。

機能

IGNORE ステートメントは、「空」のステートメントであり、それ自体は何の機能も実行しません。

アプリケーションの開発段階で、1つ以上のステートメントが必要であるが、後からコーディングするつもりでいるようなステートメントブロック内（例：AT BREAK や AT START OF DATA/AT END OF DATA 内）に、一時的に IGNORE を挿入することができます。 これにより、不完全なステートメントブロックのままでも、エラーを起こすことなくアプリケーションの他の部分を開発できます。

条件に合うときには、どの機能も実行しない場合、IF または DECIDE FOR と同様に、条件ステートメントに IGNORE ステートメントを使用する必要があります。

例

```
...
...
AT TOP OF PAGE
   IGNORE         /* top-of-page processing still to be coded
END-TOPPAGE
...
...
```

74 INCLUDE

```
INCLUDE copycode-name [operand1 ... 99]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

INCLUDE ステートメントは、コピーコードタイプの外部オブジェクトのソース行を別のオブジェクトにコンパイル時に取り込みます。

INCLUDE ステートメントはコンパイル時に評価されます。コピーコードのソース行は、INCLUDE ステートメントを含むプログラムのソース行に、物理的に組み込まれるわけではありません。プログラムのコンパイル時に組み込まれた結果が、オブジェクトモジュールとなります。

Caution: INCLUDE ステートメントを含むソースコード行に、他のステートメントは指定できません。

構文説明

オペランド定義テーブル：

オペランド	構文要素	フォーマット	ステートメント参照	ダイナミック定義
operand1	C	A U	不可	不可

構文要素の説明：

copycode-name	*copycode-name* には、挿入するソースを持つコピーコードの名前を指定します。 *copycode-name* にはアンパサンド（&）を含むことができます。この文字はコンパイル時に Natural システム変数 *LANGUAGE の現在の値で置き換えられます。 この機能で複数言語のコピーコード名を使用できます。 指定するオブジェクトは、コピーコードタイプにする必要があります。 コピーコードは、INCLUDE ステートメントを持つプログラムと同一ライブラリ、またはSTEPLIB（デフォルトの STEPLIB は SYSTEM）に保存します。 コピーコードのソースが変更された場合、その変更をオブジェクトコードに反映させるには、変更したコピーコードを使用する全プログラムを再コンパイルする必要があります。

	コピーコードのソースコードは構文的に完全なステートメトで構成されている必要があります。
operand1	組み込むコピーコードの値をダイナミックに挿入できます。 この値は *operand1* で指定します。 コピーコードでは、値は次の表記で参照されます。 `&n&` つまり、値が挿入される位置を `&n&` で指定します。 *n* は INCLUDE ステートメントで渡される各値の順番号です。 例えば、`&3&` 指定は、ステートメントで指定された 3 番目の値を参照します。 コピーコード内の各 `&n&` に対して、INCLUDE ステートメントに値を指定する必要があります。 例えば、コピーコードが `&5&` を含んでいる場合、*operand1* は少なくとも 5 回指定する必要があります。 1つのコピーコードパラメータ（`&n&`）を、空白を入れずに別のコピーコードパラメータの後に指定することができます（`&1&&2&&3&`）。 この方法は、複数のコピーコードパラメータをソースへ連結するために使用されます。 文字列を、空白を入れずに1つまたは複数のコピーコードパラメータに続けることができます（`&1&abc` または `&1&&2&abc`）。 この方法は、複数のコピーコードパラメータに文字列を連結するために使用されます。 **注意:** `&n&` は識別子の有効な部分であるため、この表記法は、上記の他の位置でコピーコードパラメータ代用（`abc&1&` または `&1&abc&2&`）として使用することはできません。 つまり、文字列は、コピーコードパラメータの後にのみ指定可能で、前または間に指定することはできません。 INCLUDE ステートメントに値を指定してもコピーコード内で参照しなければ無視されます。

例

- 例 1 - コピーコードを含む INCLUDE ステートメント
- 例 2 - パラメータのあるコピーコードを含む INCLUDE ステートメント
- 例 3 - ネスト構造のコピーコードを使用した INCLUDE ステートメント

例 1 - コピーコードを含む INCLUDE ステートメント

INCLUDE ステートメントを含むプログラム：

```
** Example 'INCEX1': INCLUDE  (include copycode)
************************************************************************
*
WRITE 'Before copycode'
*
INCLUDE INCEX1C
*
WRITE 'After copycode'
*
END
```

含めるコピーコード **INCEX1C**：

```
** Example 'INCEX1C': INCLUDE (copycode used by INCEX1)
************************************************************************
*
WRITE 'Inside copycode'
```

プログラム **INCEX1** の出力：

```
Page      1                                                 05-01-25  16:26:36

Before copycode
Inside copycode
After copycode
```

例 2 - パラメータのあるコピーコードを含む INCLUDE ステートメント

INCLUDE ステートメントを含むプログラム **INCEX2**：

```
** Example 'INCEX2': INCLUDE  (include copycode with parameters)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
*
INCLUDE INCEX2C 'EMPL-VIEW' 'NAME' '''ARCHER''' '20' '''BAILLET'''
```

```
*
END
```

含めるコピーコード **INCEX2C**：

```
** Example 'INCEX2C': INCLUDE (copycode used by INCEX2)
************************************************************************
* Transferred parameters from INCEX2:
*
* &1& : EMPL-VIEW
* &2& : NAME
* &3& : 'ARCHER'
* &4& : 20
* &5& : 'BAILLET'
*
*
READ (&4&) &1&  BY &2&  = &3&
  DISPLAY &2&
  IF &2& = &5&
    WRITE 5X 'LAST RECORD FOUND' &2&
    STOP
  END-IF
END-READ
*
* Statements above will be completed to:
*
* READ (20) EMPL-VIEW  BY NAME  = 'ARCHER'
*   DISPLAY NAME
*   IF NAME = 'BAILLET'
*     WRITE 5X 'LAST RECORD FOUND' NAME
*     STOP
*   END-IF
* END-READ
```

プログラム **INCEX2** の出力：

```
Page      1                                                 05-01-25  16:30:43

        NAME
--------------------

ARCHER
ARCONADA
ARCONADA
ARNOLD
ASTIER
ATHERTON
ATHERTON
ATHERTON
```

```
AUBERT
BACHMANN
BAECKER
BAECKER
BAGAZJA
BAILLET
     LAST RECORD FOUND BAILLET
```

例 3 - ネスト構造のコピーコードを使用した INCLUDE ステートメント

INCLUDE ステートメントを含むプログラム：

```
** Example 'INCEX3': INCLUDE  (using nested copycodes)
************************************************************************
DEFINE DATA LOCAL
1 #A (I4)
END-DEFINE
*
MOVE 123 TO #A
WRITE 'Program  INCEX3  ' '=' #A
*
INCLUDE INCEX31C '#A' '5'            /* source line is #A := 5
*
*
MOVE 300 TO #A
WRITE 'Program  INCEX3  ' '=' #A
*
INCLUDE INCEX32C '''#A'''  '''20'''  /* source line is #A := 20
*
WRITE 'Program  INCEX3  ' '=' #A
END
```

含めるコピーコード **INCEX31C**：

```
** Example 'INCEX31C': INCLUDE (copycode used by INCEX3)
************************************************************************
* Transferred parameters from INCEX3:
*
* &1& : #A
* &2& : 5
*
*
&1& := &2&
*
WRITE 'Copycode INCEX31C' '=' &1&
```

含めるコピーコード **INCEX32C**：

```
** Example 'INCEX32C': INCLUDE (copycode used by INCEX3)
************************************************************************
* Transferred parameters from INCEX3:
*
* &1& : '#A'
* &2& : '20'
*
*
WRITE 'Copycode INCEX32C' &1& &2&
*
INCLUDE INCEX31C &1& &2&
```

プログラム **INCEX3** の出力：

```
Page      1                                                 05-01-25  16:35:36

Program  INCEX3   #A:         123
Copycode INCEX31C #A:           5
Program  INCEX3   #A:         300
Copycode INCEX32C #A 20
Copycode INCEX31C #A:          20
Program  INCEX3   #A:          20
```

例 4 - コピーコードに連結パラメータを含む INCLUDE ステートメント

INCLUDE ステートメントを含むプログラム：

```
** Example 'INCEX4': INCLUDE  (with concatenated parameters in copycode)
************************************************************************
DEFINE DATA LOCAL
1 #GROUP
  2  ABC(A10) INIT <'1234567890'>
END-DEFINE
*
INCLUDE INCEX4C  '#GROUP.'  'ABC'  'AB'
*
END
```

含めるコピーコード **INCEX4C**：

```
** Example 'INCEX4C': INCLUDE (copycode used by INCEX4)
************************************************************************
* Transferred parameters from INCEX4:
*
* &1& : #GROUP.
* &2& : ABC
* &3& : AB
*
*
WRITE  '=' &2&          /* 'ABC'              results into ABC
WRITE  '=' &1&ABC       /* '#GROUP.' ABC      results into #GROUP.ABC
WRITE  '=' &1&&2&       /* '#GROUP.' 'ABC'    results into #GROUP.ABC
WRITE  '=' &1&&3&C      /* '#GROUP.' 'AB' C   results into #GROUP.ABC
```

プログラム **INCEX4** の出力：

```
Page      1                                                 05-01-25  16:37:59

ABC: 1234567890
ABC: 1234567890
ABC: 1234567890
ABC: 1234567890
```

75 INPUT

このchapterでは、次のトピックについて説明します。

構文については別途説明します。 以下を参照してください。

- **INPUT 構文 1 -** ダイナミック画面レイアウトの指定
- **INPUT 構文 2 -** 定義済みマップレイアウトの使用

関連ステートメント：**DEFINE WINDOW** | **REINPUT** | **SET WINDOW**

関連機能グループ：「*対話型処理用の画面生成*」

機能

INPUT ステートメントは、会話型モードの場合、データ入力のための形式化された画面やマップを作成するために使用します。

Natural スタック（STACK ステートメント参照）と結合して使用することもできます。また、メインフレーム環境では、バッチモードで実行しているプログラムに対するユーザーデータを提供することもできます。

Natural RPC については、『*Natural リモートプロシージャコール（RPC）*』ドキュメントの「*サーバーに対する Natural ステートメントの注意事項*」を参照してください。

入力モード

INPUT ステートメントは、スクリーン、フォームまたはキーワード／デリミタの各モードで使用できます。 スクリーンモードは、通常ビデオ端末で使用します。 フォームモードは、TTY 端末で使用します。 デリミタモードは、TTY 端末およびバッチモードで使用します（メインフレームコンピュータの場合）。 デフォルトはスクリーンモードです。

セッションパラメータ IM を使用して、入力モードを変更できます。

スクリーンモード

スクリーンモードでは、INPUT ステートメントを実行すると、フィールドや位置指定に従って画面が表示されます。 画面のメッセージ行は、Natural のエラーメッセージに使用されます。 メッセージ行の位置（画面の最上段または下段）は、端末コマンド %M を使用して変更できます。 端末ユーザーは、Tab キーで個々のフィールドに位置付けることができます。

Natural には画面ウィンドウ処理が用意されています。そのため、論理画面マップのレイアウトは物理画面サイズより大きく作成できます（理論上 250 桁×250 行、ただし内部画面バッファにより制限されます）。

ウィンドウ端末コマンドの %W で、論理的および物理的ウィンドウの位置や大きさを変更できます（ウィンドウ制御の詳細は、端末コマンド %W 参照）。

物理画面に入り切らない入力フィールド（AD=A または AD=M）には、次の規則が適用されます。

- フィールドの始まりがウィンドウ内にない入力フィールドは常に保護されます。
- ウィンドウ内で始まりウィンドウ外で終わる入力フィールドは、フィールドに含まれる値がウィンドウ内に完全に表示できない場合にのみ保護されます。 この場合、フィールド長ではなく値の長さがウィンドウのサイズを超えているかどうかを決定します。充填文字（プロファイルパラメータ FC またはセッションパラメータ AD で指定）は値の一部とはみなされません。
- このように保護された入力フィールドをアクセスまたは処理するには、ウィンドウサイズを調節してフィールドや値の全体を画面表示できるようにする必要があります（端末コマンド %W を参照）。

非スクリーンモード

INPUT ステートメントは、行単位デバイスのオペレーション、またはシーケンシャルファイルからのバッチ入力処理に使用することもできます。

スクリーンモードで定義した同じマップレイアウトを、非スクリーンモードで使用することもできます。

フォームモードおよびキーワード／デリミタモードは、行モードの画面レイアウトと同様に、またはマップレイアウトなしでデータだけを処理することによって入力を処理できます。

以下の項目も参照してください。

- *非スクリーンモードでの INPUT ステートメントの使用*
- ***Natural** スタックデータの処理*

データ入力

英数字フィールドのデータは左詰めで入力する必要があります。 空白も含めてすべての文字に意味があります。 データは 1 文字ごとにバイト単位で内部フィールドに割り当てられます。 英数字フィールドに入力されるデータは、入力の際はチェックされません。

小文字および大文字の変換はパラメータ AD=T と AD=W、または端末コマンド %L と %U で制御できます。

数字フィールドのデータは、入力フィールドの中のどこでも入力できます。 リーディングおよび（または）トレーリングブランク、リーディングゼロ、リーディングサインおよび1つの小数点を使用できます。 Natural は、フィールドの内部定義に従って、値の桁合わせをします。 SG=OFF が指定されている場合は、Natural は符号桁を確保しません。 P フォーマットで定義されたフィールドのデータは、10進数形式で入力する必要があります。 必要に応じて10進がパッ

ク形式に変換されます。 全部が空白のフィールドは、値 0 とみなされます。 数字フィールドのデータは、リーディングおよび（または）トレーリングブランク、オプションで1つのリーディングサイン、オプションで1つの小数点、および数字以外のものが値に含まれていないかどうかがチェックされます。 小数点が入力されていなければ、値の右端にあると仮定します。

バイナリフィールドに対するデータは、入力フィールドのすべての桁に対して入力する必要があります（1 バイトに対して 2 桁が対応します）。 16 進数文字（0～9、A～F）だけが使用できます。 空白文字（ASCII では H'20'、または EBCDIC では H'40'）は有効で、バイナリゼロに変換されます。 バイナリフィールドのデータは、16 進文字かどうかチェックされます。

フォーマット L のフィールドに対するデータには、空白（"偽"）または空白以外（"真"）を入力できます。

フォーマット F、D および T に対するデータは、F、D および T の定数の定められた規約に従って入力します。

数値編集マスクフリーモード

フィールド要素内に、フィールド内容の表現を編集マスクでフォーマットすることができます。編集マスクは次の 2 つの目的に使用されます。

- 画面にフィールド表示するためのレイアウトの構築。
- 文字列が修正され、Enter が押されたときに、入力された文字列からフィールドデータを抽出するため。

入力される新しいデータ値は編集マスクのフォーマットと完全に一致している必要があるため、追加の挿入文字付きで表示されるフィールドデータのフォーマットを改善する利点は実際には不都合である可能性があります。

例：

```
SET GLOBALS ID=; DC=,
RESET N (N7,3)
INPUT N (AD=M EM=Z'.'ZZZ'.'ZZZ,999EUR)
END
```

出力値	表示	入力値	必要な入力	エラーになる入力
0	,000EUR	1	1,000EUR	1 1EUR 01,000EUR
1234	1.234,000EUR	1234567	1.234.567,000EUR	1234567 1.234.567 1.234.567EUR
0,123	,123EUR	1,234	1,234EUR	1,234

編集マスクを持つ数値フィールドを入力するためのもう1つのオプションは、編集マスクフリーモードと呼ばれる代替 INPUT モードを使用することです。（プロファイルパラメータ EMFM 指定のセッション起動時、または端末コマンド %FM+ により Natural セッション実行中に）アクティブにされると、編集マスク挿入文字のすべてまたは一部が入力から除外されることがあります。

ただし、挿入文字の連続文字列が編集マスクに（以下の例の "EUR" のように）表示されると、文字列を完全に供給するか、または除外することのみが可能となります。 供給されるオプションまたは必須の桁数（編集マスク文字 "Z" および "9"）には影響しません。

例：編集マスクフリーモードを有効にする

```
SET GLOBALS ID=; DC=,
SET CONTROL 'FM+'          /* activate numeric Edit Mask Free Mode
RESET N (N7,3)
INPUT N (AD=M EM=Z'.'ZZZ'.'ZZZ,999EUR)
END
```

入力値	可能な入力	エラーになる入力
1	1 1,0 001 1,00EUR 0.001 1,EUR	1EUR
1234567	1234567 1.234.567 1234.567 1234567,0 1.234.567,0 1.234.567,EUR 1.234.567,0EUR 1.234.567,000EUR	1.234.567EUR 1.234.567,000EUR
1,234	1,234 1,234EUR 001,234 0.001,234EUR 00001,234EUR	1,234EU

Note: 編集マスクフリーモードは INPUT にのみ適用され、MOVE EDITED ステートメントでは無視されます。

SB - 選択ボックス

INPUT ステートメントの選択ボックスは、メインフレームコンピュータでのみ利用可能です。Windows では、選択ボックスはマップエディタでのみ定義できます。 UNIX および OpenVMS では、選択ボックスは定義できず、Windows またはメインフレーム環境からインポートされた場合には無視されます。

選択ボックスは入力フィールドに付加することができます。 選択ボックスはコーディングをプログラム内で直接行うことができるため、フィールドにヘルプルーチンを添付する方法に代わる最適な選択肢です。 ヘルプルーチンと同様に、特別なプログラムは必要ありません。

詳細については、『*パラメータリファレンス*』でセッションパラメータ SB を参照してください。

エラー修正

INPUT フィールドに入力した値がフォーマットやフィールドの編集マスクと一致しない場合、（プログラムの実行を終了せずに）エラーメッセージが表示され、エラーのあるフィールドにカーソルが位置付けられます。 そこでユーザーは正しい値を入力でき、その後処理は継続されます。

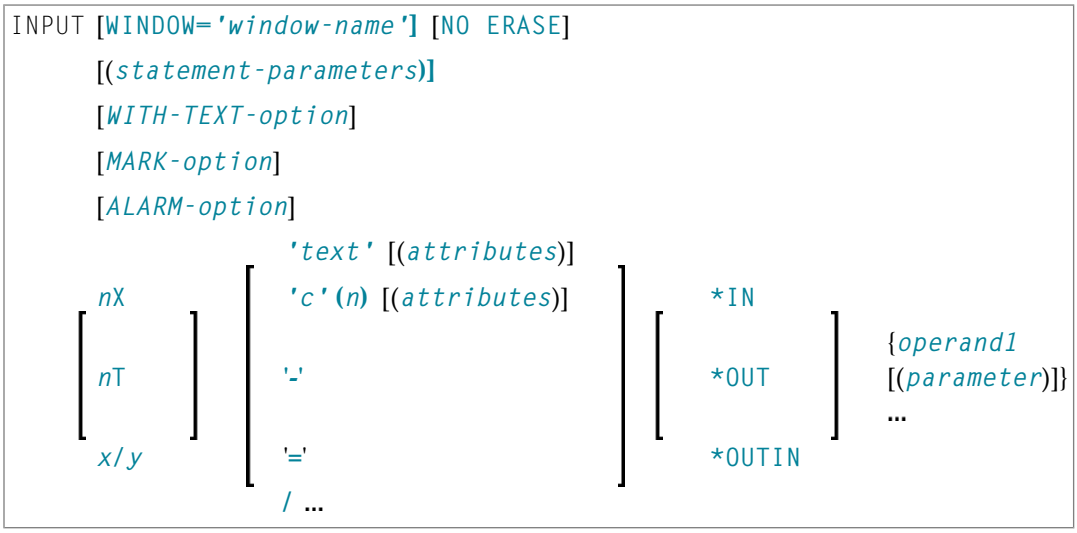
画面分割

一般的に、各 INPUT ステートメントは、新しいページ（または端末画面）を生成します。 AT END OF PAGE ステートメント内に指定された INPUT ステートメントは、新しい画面を生成しません。 この機能を使用することにより分割画面が作成できます。分割画面の上部には複数の行を表示し、画面の下部は会話のため、入力マップの作成に使用することができます。 PS パラメータ（ページサイズ）を、SET GLOBALS または FORMAT ステートメントで指定し、同一物理画面に入力マップが作成できるように論理ページサイズを設定します。

最初の INPUT 行は最終表示行の後に位置付けられます。 だたし NO ERASE オプションを使用した場合、最初の INPUT 行はページの先頭に位置付けられます。

76 INPUT構文1-ダイナミック画面レイアウトの指定

この形式の INPUT ステートメントは、INPUT 画面のレイアウトの作成、またはシーケンシャル入力ファイルからバッチモードで（メインフレームコンピュータで）読み取る INPUT データレイアウトの作成に使用されます。



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

INPUT 構文 1 - 説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1		S	A	G	N	A	U	N	P	I	F	B	D	T	L	G	可	可

構文要素の説明：

INPUT **WINDOW='*window-name*'**	オプション WINDOW='*window-name*' では、INPUT ステートメントを指定されたウィンドウに対して実行することを指定します。 指定されたウィンドウは DEFINE WINDOW ステートメントで定義されている必要があります。 指定されたウィンドウは、INPUT ステートメントの期間中にのみアクティブになり、INPUT ステートメントが実行されると自動的に非アクティブになります。 ステートメント「DEFINE WINDOW」および「SET WINDOW」も参照してください。
NO ERASE	NO ERASE を指定すると、INPUT ステートメントの画面マップが既存の画面の上にオーバーレイされます。画面の内容は消去されません。

	ここでの画面とは、物理画面ではなく論理画面を意味します。 画面に存在する保護されていないすべてのフィールドは、保護（表示専用）フィールドに変換されます。 古いデータは、新しいレイアウトが表示されるまで画面に残ります。 新しい画面の内容のフィールドが既存のフィールドを部分的にオーバーレイする場合は、新しいフィールドの前の1文字と、既存のフィールド内の次の文字が空白で置換されます。
statement-parameters	1つ以上のパラメータを、カッコで囲んで INPUT ステートメントまたは表示項目の直後に指定できます。 INPUT ステートメントで指定できるパラメータのリストについては、下記の「ステートメントのパラメータ」を参照してください。 この方法で指定した各パラメータは、その前に指定されたGLOBALSコマンド、SET GLOBALS または FORMAT ステートメントのパラメータを上書きします。 複数のパラメータを指定する場合は、各エントリ間に1つ以上の空白を配置する必要があります。 エントリを2行のステートメント行に分割することはできません。 ここで適用されるパラメータ設定は、変数フィールドにのみ関連し、テキスト定数には影響しません。 テキスト定数にフィールド属性を設定する場合は、この要素に属性を明示的に設定する必要があります。 例： `DEFINE DATA LOCAL` `1 VARI (A4)  INIT <'1234'>  /*  Output` `END-DEFINE  /*  Produced` `*  /*  ---------` `INPUT  'Text'  VARI  /*  Text 1234` `INPUT (AD=U)  'Text'  VARI  /*  Text 1234` `INPUT  'Text' (AD=U)  VARI (AD=U)  /*  Text 1234` `INPUT  'Text' (AD=U)  VARI  /*  Text 1234` `END` ステートメントレベルおよび要素レベルでのパラメータの使用例を次に示します。
WITH TEXT-option	WITH TEXT は、メッセージ行に表示されるテキストを提供するために使用します（下記の対応するセクションを参照）。
MARK-option	MARK POSITION を使用して、フィールド内の特定の位置にカーソルを配置できます。 下記の「***MARK オプション***」を参照してください。
ALARM-option	このオプションは、INPUT ステートメントの実行時にアクティブ化される端末の音声アラーム機能を提供します。 下記の「***Alarm オプション***」を参照してください。

その他の構文要素（*n*X、*n*T、*x*/*y*、*operand1* など）	下記の「*フィールドの位置、文字列、属性の指定*」を参照してください。

ステートメントのパラメータ

INPUT ステートメントで指定可能なパラメータ		**指定（S = ステートメントレベル、E = 要素レベル）**
AD	属性定義	SE
AL	出力の英数字長	SE
CD	カラー定義	SE
CV	制御変数	SE
DF	日付フォーマット	SE
DL	出力の表示長	SE
DY	ダイナミック属性	SE
EM	編集マスク	SE
EMU	Unicode 編集マスク	E
FL	浮動小数点仮数長	SE
HE	ヘルプルーチン	SE
IP	入力プロンプトのテキスト	SE
LS	行サイズ	S
MC	マルチプルバリューフィールド数	S
MS	手動による省略	S
NL	出力の数値長	SE
PC	ピリオディックグループ数	S
PM	出力モード *	SE
PS	ページサイズ **	S
SB	選択ボックス	E
SG	符号の位置	SE
ZP	ゼロ出力	SE

* PM セッションパラメータは、テキスト定数には指定できません。

** PS セッションパラメータ設定は、配列のオカレンス数が PS 値を超える場合は考慮されません。

各セッションパラメータの詳細については、『パラメータリファレンス』を参照してください。

WITH TEXT オプション

```
[WITH] TEXT [ * operand1 | operand2 ] [(attributes)][,operand3] ... 7
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S					N	P	I		B*						可	可
operand2	C	S				A											可	可
operand3	C	S				A	N	P	I	F	B	D	T	L			可	可

**operand1* のフォーマット B は、4 以下の長さでのみ使用できます。

WITH TEXT は、メッセージ行に表示されるテキストを提供するために使用します。 これは通常、画面処理やエラー修正のために実行する必要がある操作を示すメッセージです。

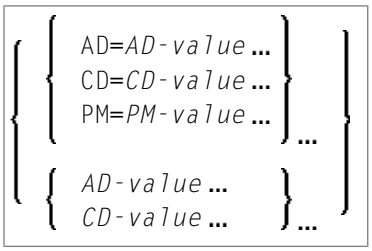
構文要素の説明：

operand1	**Natural** メッセージファイルからのメッセージテキスト： *operand1* は、Natural メッセージファイルから取得するメッセージテキストの番号を表します。 ユーザー定義メッセージまたは Natural システムメッセージのいずれかを取得できます。 ■ 最大4桁の正の値（954など）を指定すると、ユーザー定義メッセージを取得できます。 ■ 最大 4 桁の負の値（-954 など）を指定すると、Natural システムメッセージを取得できます。 REINPUT ステートメントの説明にある「*例 4 - WITH TEXT オプション*」も参照してください。 Natural メッセージファイルは、関連ドキュメントで説明しているように、SYSERR ユーティリティで作成およびメンテナンスします。
operand2	メッセージテキスト： *operand2* は、メッセージ行に配置されるメッセージを表します。 REINPUT ステートメントの説明にある「*例 4 - WITH TEXT オプション*」も参照してください。
attributes	*operand1/2* にはさまざまな出力属性を割り当てることができます。 これらの属性と構文については、以下の「*出力属性*」で説明します。
operand3	メッセージテキストのダイナミック置換： *Operand3* は、英数字、テキスト定数、または変数名を表します。

	提供された値は、*operand1* または *operand2* のいずれかで指定されるメッセージテキストの一部を置き換えるために使用されます。 :*n*: は *operand3* の内容を参照するためにメッセージテキスト内に指定します。*n* には *operand3* のオカレンス（1～7）を指定します。 REINPUT ステートメントの説明にある「*例 4 - WITH TEXT オプション*」も参照してください。 **注意:** 複数の *operand3* を指定する場合は、それぞれをコンマで区切る必要があります。コンマを小数点文字として（セッションパラメータ DC で定義）使用し、数値定数を *operand3* として指定している場合は、コンマの前後に空白を挿入して、コンマが小数点文字と解釈されないようにします。または、複数の *operand3* を INPUT 区切り文字（セッションパラメータ ID で指定）で区切ることができます。ただし、INPUT ステートメント構文ではスラッシュは別の意味を持つため、ID=/（スラッシュ）の場合は使用できません。 先頭のゼロまたは末尾の空白は、メッセージに表示される前にフィールド値から削除されます。

出力属性

attributes は、テキスト表示に使用される出力属性を示します。 可能な属性：



指定可能なセッションパラメータ値については、『パラメータリファレンス』ドキュメントの該当するセクションを参照してください。

- 「*AD - 属性定義*」の「*フィールド表現*」
- *CD - カラー定義*
- *PM - 出力モード*

Note: コンパイラは、実際には 1 つの出力フィールドに複数の属性値を受け入れます。 例えば、「AD=BDI」と指定できます。 ただし、この場合は最後の値のみが適用されます。 示した例では、値 "I" のみが有効になり、出力フィールドは強調表示されます。

MARK オプション

MARK POSITION を使用して、フィールド内の特定の位置にカーソルを配置できます。

```
[MARK POSITION operand4 [IN]] [FIELD] { operand1 }
                                      { *fieldname }
```

オペランド定義テーブル：

オペランド	構文要素						フォーマット										ステートメント参照	ダイナミック定義
operand4	C	S					N	P	I								可	可
operand1	C	S	A				N	P	I								可	可

構文要素の説明：

MARK ******fieldname*	******fieldname* は、フィールド名を使用して位置付けるために使用します。
MARK POSITION *operand4*	MARK POSITION では、*operand1* で指定したフィールド内の *operand4* で指定した特定の位置にカーソルを置くことができます。 *operand4* に小数桁を含めることはできません。 例： `MARK 3` `MARK #A` `MARK *#` `MARK POSITION 3 IN #A`
operand1	*operand1* では、マークするフィールドを指定します。 INPUT ステートメントで指定された各フィールド属性 AD=Aまたは AD=M（保護されていないフィールド）には、1 から開始するフィールド参照番号が割り当てられます。 MARK オプションは、指定されたフィールド番号にカーソルを配置します。

ALARM オプション

このオプションは、INPUT ステートメントの実行時にアクティブ化される端末の音声アラーム機能を提供します。 この機能を使用するには、対応するハードウェアが使用可能である必要があります。

```
[[AND] [SOUND] ALARM]
```

デフォルトのプロンプトテキスト

セッションパラメータ IP（入力プロンプト）が IP=OFF に設定されていない限り、INPUT ステートメントで使用されるフィールドのフィールド名が、フィールド値の前に（フォームモード）、またはフィールドを選択するためのプロンプトキーワードとして（キーワード／デリミタモード）表示されます。 このデフォルトのフィールド名は、'*text*' 要素（デフォルト名を置換）または '-'（デフォルトフィールド名の表示を抑制）をフィールド名の直前に指定することで上書きできます。

フィールドの位置、文字列、属性の指定

```
[ nX  ]   [ 'text'  [(attributes)]  ]   [ *IN    ]
[ nT  ]   [ 'c'(n) [(attributes)]   ]   [ *OUT   ]   {operand1
[ x/y ]   [        '-'              ]   [ *OUTIN ]   [(parameter(s))]}
          [        '='              ]
          [        / ...            ]
```

フィールドの位置、属性の割り当て、およびテキストの作成には複数の表記を使用できます。

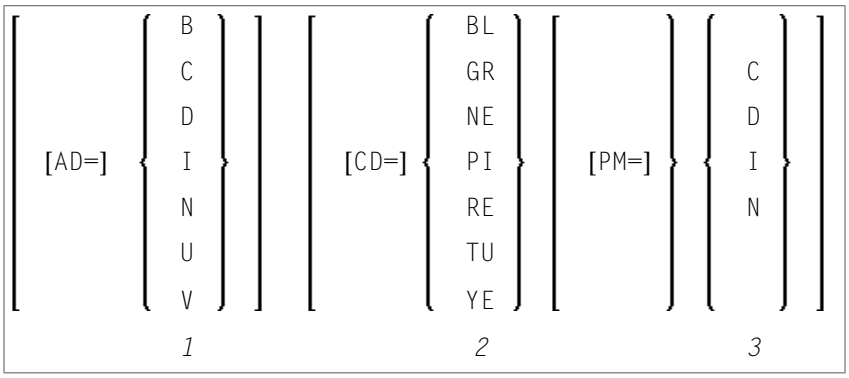
***n*X**	*n* 個のスペースがフィールド間に挿入されます。
***n*T**	位置 *n* を出力するように位置指定（タブ設定）されます。
x/*y*	次の要素を行 *x* の列 *y* から開始して配置します。 *y* を "0" にすることはできません。 同じ行で後方に位置指定することはできません。
'*text*'	*text*が書き込み保護状態で表示されます。「*テキスト表記*」の「*ステートメントで使用するテキストの定義*」も参照してください。
'*c*' (*n*)	'*text*' と同一ですが、文字 *c* が *n* 回表示されます。 *n* は 1～132 にする必要があります。「*テキスト表記*」」の「*フィールド値の前に n 回表示する文字の定義*」も参照してください。
attributes	表示に使用される属性を示します。 下記の「*属性*」を参照してください。

'-'	マイナス記号： フィールドの前に配置された場合、'-' はプロンプトテキストとしてのフィールド名の生成を抑制します。 **注意:** フィールドの前のテキスト文字列は、フィールド名をプロンプトテキストとして置換します。
'='	等号： フィールドの前に '=' を配置した場合は、後にフィールドの内容が続くフィールドヘッダーが表示されます。
'/'	スラッシュ記号： フィールドまたはテキスト要素の間に '/' を配置した場合は、次の出力行の先頭に位置指定されます。 フィールドの内容は、属性設定 AD=A、AD=O および AD=M を使用して、それぞれ入力、出力のみ、変更用の出力に指定できます。デフォルトは AD=A です。 AD=A（入力のみ）または AD=M（変更用の出力）が指定されたすべてのフィールドでは、保護されていないフィールドが画面に作成されます。このようなフィールドの値はユーザーが入力できます。TTY デバイスの場合、変更用の出力フィールドは、新しい値を入力できるように、フィールドのサイズの 2 倍（出力用に 1 つと入力用に1つ）を占めます。表示不能として指定された入力フィールド（AD=A/M）は、TTY デバイスで常に新しい行で開始します。 例： `INPUT #A (AD=A) #B (AD=O) #C (AD=M)` #A は、保護されていない入力フィールドです。つまり、フィールドに値が入力されます。 #B は、書き込み保護で表示されるフィールドです。つまり、フィールドには値を入力できません。 #C は、現在の値が表示されるフィールドであり、フィールドに新しい値を入力することで値を変更できます。
***IN、*OUT および *OUTIN**	それぞれ属性 AD=A、AD=O、AD=M と同等です。 **注意:** 変更不能なシステム変数が INPUT ステートメントで使用されている場合は、値が出力専用フィールド AD=O または *OUT 属性として表示されます。

operand1	フィールド指定： *operand1* は、使用するフィールドを表します。 データベースフィールドまたはユーザー定義変数を指定できます。 Natural は、データエリアの各フィールドの内容を INPUT ステートメントにマップするため、移動操作は不要です。 データベースフィールドの内容が INPUT 処理の結果として変更される場合は、データエリアにある値のみが変更されます。 適切なデータベースステートメント UPDATE/STORE を使用して、データベースの内容を変更する必要があります。 データベースフィールドのグループの名前が INPUT ステートメントで参照されている場合は、そのグループに属するすべてのフィールドが入力フィールドとして個別に使用されます。 配列内のオカレンスの範囲に対する参照が行われた場合は、すべてのオカレンスが個々に入力フィールドとして処理されますが、プロンプトテキストは個々のオカレンスに対しては作成されず、最初のオカレンスに対してのみ作成されます。 メインフレームコンピュータでは、実行時にオカレンス数を変更できる、範囲による配列は指定できません。
parameter(s)	1 つまたは複数のパラメータをカッコで囲んで *operand1* の直後に指定できます（下記の表と例を参照）。 指定した各パラメータは、その前に指定された GLOBALS コマンド、SET GLOBALS（レポーティングモード）または FORMAT ステートメントのパラメータを上書きします。 複数のパラメータを指定する場合は、個々のパラメータを 1 つ以上の空白で区切る必要があります。 各パラメータ指定を 2 行のステートメント行に分割することはできません。 ここで適用されるパラメータ設定は、変数フィールドにのみ関連し、テキスト定数には影響しません。 テキスト定数にフィールド属性を設定する場合は、この要素に属性を明示的に設定する必要があります。 個々のパラメータの詳細については、「*ステートメントのパラメータ*」の表を参照してください。 **注意:** データベースフィールドに編集マスクが定義されている場合は、セッションパラメータ EM が DDM で動的に参照されます。 編集マスクは出力フィールドと入力フィールドに対してのみ指定できます。 編集マスクが入力フィールドに対して定義された場合、フィールドのデータは編集マスクの仕様に従って入力する必要があります。

属性

次の属性を使用できます。



1. 表示属性：『パラメータリファレンス』でセッションパラメータ AD を参照してください。
2. 色属性：『パラメータリファレンス』でセッションパラメータ CD を参照してください。
3. 出力モード属性：『パラメータリファレンス』でセッションパラメータ PM を参照してください。

例 - 構文 1

- 例 1 - INPUT ステートメント
- 例 2 - DEFINE WINDOW ステートメントのある INPUT ステートメント
- 例 3 - MARK POSITION オプションのある INPUT ステートメント

例 1 - INPUT ステートメント

```
** Example 'IPTEX1': INPUT
************************************************************************
DEFINE DATA LOCAL
1 #FNC  (A1)
END-DEFINE
*
INPUT 10X 'SELECTION MENU FOR EMPLOYEES SYSTEM' /
      10X '-' (35) //
      10X 'ADD      (A)' /
      10X 'UPDATE   (U)' /
      10X 'DELETE   (D)' /
      10X 'STOP     (.)' //
      10X 'PLEASE ENTER FUNCTION: ' #FNC
*
DECIDE ON EVERY VALUE OF #FNC
  VALUE 'A'   /* invoke the object containing the add function here
    WRITE 'Add function selected.'
  VALUE 'U'   /* invoke the object containing the update function here
    WRITE 'Update function selected.'
  VALUE 'D'   /* invoke the object containing the delete function here
```

```
    WRITE 'Delete function selected.'
  VALUE '.'
    STOP
  NONE
    REINPUT 'Please enter a valid function.' MARK *#FNC
END-DECIDE
*
END
```

プログラム **IPTEX1** の出力：

```

SELECTION MENU FOR EMPLOYEES SYSTEM
-----------------------------------

ADD      (A)
UPDATE   (U)
DELETE   (D)
STOP     (.)

PLEASE ENTER FUNCTION:
```

例 2 - DEFINE WINDOW ステートメントのある INPUT ステートメント

```

** Example 'INPEX1': INPUT (with DEFINE WINDOW statement)
************************************************************************
DEFINE DATA LOCAL
1 #STRING (A15)
END-DEFINE
*
DEFINE WINDOW WIND1
  SIZE 10 * 40
  BASE 5 / 10
  FRAMED ON POSITION TEXT
*
INPUT WINDOW='WIND1'
      'PLEASE ENTER HERE:' / #STRING
*
END
```

プログラム **INPEX1** の出力：

```
+----------------------------------Top+
! PLEASE ENTER HERE:                  !
! #STRING                            !
!                                    !
!                                    !
!                                    !
!                                    !
!                                    !
!                                    !
+--------------------------------Bottom+
```

例 3 - MARK POSITION オプションのある INPUT ステートメント

```
** Example 'INPEX2': INPUT (with POSITION)
************************************************************************
DEFINE DATA LOCAL
1 #START (A30)
END-DEFINE
*
ASSIGN #START = 'EXAM_'
*
INPUT (AD=M) MARK POSITION 5 IN *#START
     / 'PLEASE COMPLETE START VALUE FOR SEARCH'
     / 5X #START
END
```

プログラム **INPEX2** の出力：

```
PLEASE COMPLETE START VALUE FOR SEARCH
       #START EXAM[]
```

77 INPUT 構文 2 - 定義済みマップレイアウトの使用

この形式の INPUT ステートメントは、Natural マップエディタを使用して作成されたマップレイアウトを使用して入力処理を実行するために使用されます。

マップレイアウトは次の 2 つの方法で使用できます。

- プログラムがパラメータリストを提供しない。
- プログラムがパラメータリスト（*operand1*）を提供する。

```
INPUT [WINDOW='window-name'] [WITH-TEXT-option]
 [MARK-option]
 [ALARM-option]
 [USING] MAP map-name [NO ERASE]
[                operand1 ...     ]
[                NO PARAMETER     ]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

パラメータリストのない INPUT USING MAP

パラメータリストのない INPUT USING MAP を使用する場合は、次の要件を満たす必要があります。

- *map-name* を英数字定数として指定する必要があります（最大 8 文字）。
- この方法で使用されるマップは、マップを参照するプログラムをコンパイルする前に作成する必要があります。
- 処理されるフィールドの名前は、コンパイル時にマップソース定義から動的に取得されます。使用されるフィールド名はプログラムとマップの両方で同一である必要があります。
- INPUT ステートメントで参照されるすべてのフィールドは、その時点でアクセス可能である必要があります。
- ストラクチャードモードでは、フィールドはあらかじめ定義されている必要があります（データベースフィールドは、処理ループまたはビューに正しく参照される必要があります）。
- レポーティングモードでは、マップにユーザー定義変数を新たに定義できます。
- マップレイアウトが変更された場合でも、マップを使用しているプログラムを再カタログする必要がありません。 ただし、配列構造または名前、フィールドの形式／長さが変更された場合、またはマップでフィールドが追加／削除された場合は、マップを使用しているプログラムを再カタログする必要があります。
- マップソースはプログラムコンパイル時に使用可能である必要があります。そうでないと、INPUT USING MAP ステートメントをコンパイルできません。

Note: マップがまだ使用可能でない場合にプログラムをコンパイルするには、NO PARAMETER を指定します。これにより、マップがまだ使用可能でなくても INPUT USING MAP をコンパイルできます。

プログラムで定義されている INPUT フィールド

プログラムで処理するフィールドの名前（*operand1*）を指定することで、マップ内のフィールドの名前とは異なる名前をプログラムのフィールドの名前にすることができます。

プログラムのフィールドの順序はマップの順序と一致している必要があります。 マップエディタは、マップで指定されたフィールドを、フィールド名の英字順でソートすることに注意してください。詳細については、Naturalの『エディタ』ドキュメントのマップエディタの説明を参照してください。

プログラムエディタの行コマンド .I(*mapname*) を使用して、指定されたマップで定義されているフィールドから派生したパラメータリストとともに完全な INPUT USING MAP ステートメントを取得できます。

マップのレイアウトが変更された場合でも、マップを使用してプログラムを再カタログする必要はありません。 ただし、フィールド名、フィールド形式／長さ、またはマップ内の配列構造が変更された場合、またはマップでフィールドが追加または削除された場合は、プログラムを再カタログする必要があります。

プログラムで指定されたフィールドの形式と長さが、マップで指定されたフィールドと一致することを確認するために、実行時にチェックが行われます。 両方のレイアウトが一致しない場合は、エラーメッセージが生成されます。

INPUT 構文 2 - 説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
map-name	C	S				A	U											可	不可
operand1		S	A			A	U	N	P	I	F	B	D	T	L	C		可	可

構文要素の説明：

INPUT WINDOW='*window-name***'**	このオプションについては、INPUT ステートメントの「*構文 1*」で説明しています。
WITH *TEXT/MARK/ALARM-options*	これらのオプションについては、INPUT ステートメントの「*構文 1*」で説明しています。「***WITH TEXT オプション***」、「***MARK オプション***」、「***ALARM オプション***」を参照してください。
USING MAP *map-name*	USING MAP は、マップエディタを使用して以前に Natural システムに格納されたマップ定義を呼び出します。 *map-name* は、1～8 文字の英数字定数またはユーザー定義変数にすることができます。変数を使用する場合、事前にこれを定義しておく必要があります。マップ名にアンパサンド（&）を含めることができます。この文字は実行時に Natural システム変数 *LANGUAGE の現在の値で置き換えられます。 この機能で複数言語のマップを使用できます。 INPUT ステートメントを実行すると、NO ERASE オプションが指定されていない限り、対応するマップで画面の現在の内容が置き換えられます。このオプションを指定した場合は、マップが画面の現在の内容をオーバーレイします。
NO ERASE	このオプションについては、INPUT ステートメントの「*構文 1*」で説明しています。
operand1	フィールド指定： データベースフィールドまたはユーザー定義変数、あるいはその両方のリスト。 フィールドは、参照先のマップ内のフィールドと、数、順序、形式、長さ、およびオカレンス数（配列の場合）が一致している必要があります。そうでないとエラーが発生します。 データベースフィールドの内容が INPUT 処理の結果として変更される場合は、データエリアにある値のみが変更されます。 適切なデータベーステートメント UPDATE/STORE を使用して、データベースの内容を変更する必要があります。

非スクリーンモードでの INPUT ステートメントの使用

セッションパラメータ IM を使用して、入力モードを変更できます。

フォームモード

フォームモード（プロファイル／セッションパラメータ IM=F）では、Naturalは位置パラメータに従ってフィールドごとに端末にマップレイアウトの出力テキストをすべて表示します。 これにより、ユーザーはフィールドごとにデータを入力できます。すべてのデータが入力されると、ハードコピーの出力が、画面に表示されるのと同じ形式で生成されます。

フォームモードでは、%R と入力すると、エラーの発生時にオペレータがフォーム全体を再入力できます。 入力は、INPUT ステートメントの最初の実行時に処理されます。

キーワード／デリミタモード

キーワード／デリミタモード（プロファイル／セッションパラメータ IM=D）では、キーワードまたは位置入力値を使用してデータを入力できます。

キーワード入力を使用すると、端末オペレータは、プロンプトテキストを使用して個々のフィールドのデータを入力できます。フォームモードでは、プロンプトテキストは、フィールドを識別するキーワードとして値の前に表示されます。 キーワードの後には、INPUT割り当て文字（IAパラメータ）、その直後にデータが続く必要があります。 割り当て文字に続くスペースは、区切り文字（IDパラメータ）までデータとして取得されます。 区切り文字は、最後のデータ要素の後には不要です。 異なるフィールドのキーワードデータは、区切り文字で分割して任意の順序で入力できます。 INPUT ステートメントで定義されていないキーワードをオペレータが入力した場合は、エラーメッセージが返されます。 入力フィールドにデータを入力する必要はありません。 データが入力されていないフィールドは、英数字フィールドでは空白に、数値および16進数フィールドではゼロに設定されます。

位置値入力を使用する場合、端末オペレータは、現在定義されているINPUT区切り文字（IDパラメータ）で区切られたすべての入力フィールドのデータのみ入力します。 入力するフィールドの順序は、INPUT ステートメントのフィールドの順序に対応します。

ユーザーは、デリミタ文字で区切った位置入力に値の数を入力し、値の前にキーワードを指定して選択したフィールドのキーワードモードに切り替えることにより、位置入力からキーワード入力に切り替えることができます。

キーワードがフィールドの位置指定に使用された場合、キーワードの値に続くキーワード以外の入力は、INPUTステートメントで以前に選択されたフィールドに続くフィールドに割り当てられる位置入力として処理されます。

Note: キーワードおよび対応する入力フィールドは、同じ論理行にある必要があります。合計の長さが行サイズを超える場合は、行サイズ（LSパラメータ）を適宜調整して、キーワードとフィールドが1行に収まるようにします。

キーワード／デリミタモードで入力されたデータは、スクリーンモードの場合と同じように検証されます。フィールドに対して定義されている文字数を超えて入力しようとした場合は、エラーメッセージが返されます。

バッファ（3270タイプ）端末またはワークステーションで、INPUT ステートメントをキーワード／デリミタモードで処理する場合は、1つの INPUT ステートメントに割り当てるすべてのデータを1つの画面に入力する必要があります。 Enter は、INPUT ステートメントに対するすべてのデータが入力された場合にのみ使用します。

Natural スタックデータの処理

Natural スタックに FETCH、RUN、または STACK ステートメントを介して配置されたデータ要素は、実行時に検出された次の INPUT ステートメントで処理されます。

INPUT ステートメントは、前述したようにキーワード／デリミタモードのデータを処理します。

すべての入力フィールドを充填するためのデータ要素が使用可能でない場合は、フィールド形式に応じて空白／ゼロが充填されます。 存在する入力フィールドよりも多くのデータ要素が指定されている場合、残りのデータは無視されます。

フィールドにスタックからのデータが入力されている場合、フィールド属性はデータに適用されません。

Natural システム変数 *DATA を参照して、Natural スタックで現在使用可能なデータ要素の数を判断できます。

78 INTERFACE

```
INTERFACE  interface-name
 [ EXTERNAL]
 [ID interface-GUID]
 [property-definition]
 [method-definition]
END-INTERFACE
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CREATE OBJECT** | **DEFINE CLASS** | **INTERFACE** | **METHOD** | **PROPERTY** | **SEND METHOD**

関連機能グループ：「*コンポーネントベースプログラミング*」

機能

インターフェイスは、互いに意味のあるメソッドおよびプロパティの集まりであり、クラスの特定機能を提供します。

クラスに対して 1 つ以上のインターフェイスを定義することができます。 複数インターフェイスを定義すると、それらが行うことに従ってメソッドを構造化／グループ化することが可能です。例えば、すべてのメソッドを挿入し、1つのインターフェイスの持続性（ロード、ストア、更新）を扱い、他のメソッドを他のインターフェイスに挿入します。

`INTERFACE` ステートメントは、インターフェイスを定義するために使用します。 Natural クラスモジュールでのみ使用し、次のように定義できます。

- `DEFINE CLASS` ステートメント内で定義します。 この形式は、インターフェイスが 1 つのクラスに実装されるときにのみ使用します。
- または、`DEFINE CLASS` ステートメントの `INTERFACE USING` 節に組み込まれるコピーコードで定義します。 この形式は、インターフェイスが複数のクラスに実装されるときに使用します。

インターフェイスに関連するプロパティとメソッドは、プロパティとメソッド定義で定義します。

構文説明

interface-name	インターフェイスに割り当てる名前です。 インターフェイス名は最大 32 文字以内で、ユーザー定義変数の 命名規則に従っている必要があります（『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照）。 クラスごとに一意で、クラス名とも異なっている必要があります。 異なるプログラミング言語で書かれたクライアントによってインターフェイスを使用する予定であれば、インターフェイス名はこれらの言語に適用する命名規則と矛盾しないように選択する必要があります。
EXTERNAL	EXTERNAL 節は、当インターフェイスがクラスによって実装されるが、本来は別のクラスで定義されていることを示します。 EXTERNAL 節は、クラスが DCOM で登録される場合に限り関連します。 EXTERNAL 節のインターフェイスは、クラスが DCOM で登録されるときには無視されます。 インターフェイスは、本来それを定義するクラスによって登録されることを前提とします。
ID *interface-GUID*	ID 節は、グローバルユニーク ID をインターフェイスに割り当てます。 インターフェイス GUID は、LOCAL 節に組み込まれるデータエリアに定義された GUID の名前です。 インターフェイス GUID は英文字定数です。 GUID は、クラスが DCOM で登録される場合にインターフェイスに割り当てる必要があります。
property-definition	プロパティ定義は、インターフェイスのプロパティを定義するために使用します。 下記の「*プロパティ定義*」を参照してください。
method-definition	メソッド定義は、インターフェイスのメソッドを定義するために使用します。 下記の「*メソッド定義*」を参照してください。
END-INTERFACE	INTERFACE ステートメントを終了するには、Natural の予約語 END-INTERFACE を使用する必要があります。

プロパティ定義

プロパティ定義は、インターフェイスのプロパティを定義するために使用します。

```
PROPERTY property-name
 [(format-length/array-definition)]
 [ID dispatch-ID]
 [READONLY]
 [IS operand]
END-PROPERTY
```

プロパティは、クライアントによってアクセス可能なオブジェクトの属性です。 例えば、従業員を表すオブジェクトは "Name" プロパティと "Department" プロパティを持ちます。 "Name" または "Department" プロパティをアクセスすることによって従業員の名前や部署を検索したり

変更したりするほうが、値を返す1つのメソッドおよび値を変更する別のメソッドを呼び出すより、クライアントにとっては非常に簡単です。

各プロパティは、その値を保存するためにクラスのオブジェクトデータエリアに変数を必要とします。これはオブジェクトデータ変数として参照されます。 プロパティ定義を使用してこの変数をクライアントにアクセス可能にします。プロパティ定義は、プロパティの名前とフォーマットを定義し、オブジェクトデータ変数に接続します。 最も単純なケースでは、プロパティは、オブジェクトデータ変数自体の名前とフォーマットを使用します。一定の制限内の名前とフォーマットを変更することも可能です。

property-name	プロパティに割り当てる名前です。 プロパティ名は最大 32 文字以内で、Natural ユーザー定義変数の命名規則に従う必要があります。『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照してください。 異なるプログラミング言語で書かれたクライアントによってプロパティを使用する予定であれば、プロパティ名はこれらの言語に適用する命名規則と矛盾しないように選択する必要があります。
format-length/array-definition	クライアントに認識されるように、プロパティのフォーマットを定義します。 *format-length/array-definition*を省略すると、フォーマット長さと配列定義は、IS 節で割り当てられたオブジェクトデータ変数から取得されます。 *format-length/array-definition* を指定する場合、IS 節の*operand*に指定されたオブジェクトデータ変数のフォーマットとの転送の互換性が必要です。 READONLY プロパティの場合、データ転送の互換性は 1 つの方向（ソースオペランドとしてのオブジェクトデータ変数と宛先オペランドとしてのプロパティ）にのみ必要です。配列定義を指定するときは、次元、次元ごとのオカレンス、下限、および上限を、対応するオブジェクトデータ変数の配列定義と等しくする必要があります。 これは、次元ごとに 1 つのアスタリスクを指定することによって表現されます。
ID *dispatch-ID*	ID 節は、特定の数値識別子をプロパティに割り当てます。 この ID（ディスパッチ ID）は、クラスを DCOM に登録する場合にのみ関連します。 通常、Natural は自動的にディスパッチ ID をプロパティに割り当てます。 プロパティが EXTERNAL 節のインターフェイスに属している場合にのみ、プロパティの固有ディスパッチ ID を明示的に定義する必要があります。（これは、このクラスで実装されるインターフェイスですが、本来は別のクラスで定義されています。）この場合、使用するディスパッチ ID は、通常インターフェイスの本来の実装によって指示されます。 ディスパッチ ID は、フォーマット I4 のゼロでない正の定数です。

READONLY	このキーワードが指定されると、プロパティの値は読まれるだけで、設定されません。 IS 節の *operand* に指定されたオブジェクトデータ変数のフォーマットは、*format-length/array-definition* に指定されたフォーマットへのデータ転送の互換性が必要です。逆方向へのデータ転送の互換性は必要ありません。 キーワード READONLY が省略されると、プロパティの値は読み込みおよび設定が可能です。
IS *operand*	IS 節の *operand* により、オブジェクトデータ変数をプロパティ値の保存場所として割り当てます。割り当てられたオブジェクトデータ変数はグループでなくてもかまいません。変数は正常なオペランド構文で参照されます。これは、オブジェクトデータ変数が配列であれば、添字指定で参照する必要があることを意味しています。完全な添字範囲指定とアスタリスク表記だけが許されます。 INTERFACE ステートメントがコピーコードメンバから組み込まれ、複数のクラスで再利用されるならば、IS 節は使用されません。 INTERFACE ステートメントを再利用したい場合は、INTERFACE ステートメント外で PROPERTY ステートメントにオブジェクトデータ変数を割り当てる必要があります。 IS 節を省略すると、プロパティはプロパティと同じ名前のオブジェクトデータ変数と接続されます。この名前を持つ変数が定義されていないか、または、それがグループであれば、構文エラーが生じます。
END-PROPERTY	Natural 予約語 END-PROPERTY を使用して、インターフェイス PROPERTY 定義を終了させる必要があります。

例

オブジェクトデータエリアに以下のデータ定義を含ませてください。

```
1 Salary(p7.2)
  1 SalaryHistory(p7.2/1:10)
```

その後、以下のプロパティ定義が可能になります。

```
property Salary
  end-property
  property Pay is Salary
  end-property
  property Pay(P7.2) is Salary
  end-property
  property Pay(N7.2) is Salary
  end-property
  property SalaryHistory
  end-property
  property OldPay is SalaryHistory(*)
  end-property
  property OldPay is SalaryHistory(1:10)
  end-property
  property OldPay(P7.2/*) is SalaryHistory(1:10)
  end-property
  property OldPay(N7.2/*) is SalaryHistory(*)
  end-property
```

以下のプロパティ定義は許されません。

```
/* Not data transfer-compatible. */
  property Pay(L) is Salary
  end-property
  /* Not data transfer-compatible. */
  property OldPay(L/*) is SalaryHistory(*)
  end-property
  /* Not data transfer-compatible. */
  property OldPay(L/1:10) is SalaryHistory(1:10)
  end-property
  /* Assigns an array to a scalar. */
  property OldPay(P7.2) is SalaryHistory(1:10)
  end-property
  /* Takes only a sub-array. */
  property OldPay(P7.2/3:5) is SalaryHistory(*)
  end-property
  /* Index specification omitted in ODA variable SalaryHistory. */
  property OldPay is SalaryHistory
  end-property
  /* Only asterisk notation allowed in property format specification. */
```

```
property OldPay(P7.2/1:10) is SalaryHistory(*)
end-property
```

メソッド定義

メソッド定義は、インターフェイスのメソッドを定義するために使用します。

```
METHOD method-name
 [ID dispatch-ID]
 [IS subprogram-name]
[   PARAMETER { USING parameter-data-area }   ]...
              { data-definition...        }
END-METHOD
```

インターフェイスを各種クラスで再利用可能にするためには、コピーコードからインターフェイス定義を組み込み、インターフェイス定義の後に METHOD ステートメントでサブプログラムを定義してください。 その後、各種クラスでそれぞれにメソッドを実装できます。

method-name	メソッドに割り当てる名前です。 メソッド名は最大 32 文字で、Natural の命名規則に従う必要があります（『*Natural スタジオの使用*』ドキュメントの「*ユーザー定義変数の命名規則*」を参照）。 インターフェイスごとに一意にする必要があります。 異なるプログラミング言語で書かれたクライアントによってメソッドを使用する予定であれば、メソッド名はこれらの言語に適用する命名規則と矛盾しないように選択する必要があります。
ID *dispatch-ID*	ID節は、特定の数値識別子をメソッドに割り当てます。このID（ディスパッチID）は、クラスを DCOM に登録する場合にのみ関連します。 通常、Natural は自動的にディスパッチ ID をメソッドに割り当てます。 メソッドが EXTERNAL 節のインターフェイスに属している場合にのみ、メソッドの固有ディスパッチIDを明示的に定義する必要があります。（これは、このクラスで実装されるインターフェイスですが、本来は別のクラス定義されています。）この場合、使用するディスパッチ ID は、通常インターフェイスの本来の実装によって指示されます。 ディスパッチ ID は、フォーマット I4 のゼロでない正の定数です。
IS *subprogram-name*	これは、メソッドを実装するサブプログラムの名前です。 サブプログラムの名前は最大 8 文字です。 デフォルトはメソッド名です（IS 節が指定されない場合）。
PARAMETER	メソッドのパラメータを指定します。DEFINE DATA ステートメントの PARAMETER 節と同じ構文です。 パラメータは、サブプログラムの実装において後で使用されるパラメータと一致している必要があります。 これは、パラメータデータエリアを使用することによって保証されます。

	パラメータデータエリアの BY VALUE とマークされたパラメータは、メソッドの入力パラメータです。 BY VALUE が指定されていないパラメータは、「by reference」（参照）で渡される入力／出力パラメータです。 これがデフォルトです。 BY VALUE RESULT とマークされている最初のパラメータが、メソッドの戻り値として返されます。 複数のパラメータがこのようにマークされている場合は、その他のパラメータは入力／出力パラメータとして扱われます。
END-METHOD	Natural 予約語 END-METHOD を使用して、インターフェイスの METHOD 定義を終了させる必要があります。

79 LIMIT

```
LIMIT n
```

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION** | **HISTOGRAM** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

LIMIT ステートメントは、FIND、READ または HISTOGRAM ステートメントで開始された処理ループの実行回数を制限するために使用します。

制限値は、他の LIMIT ステートメントで上書きされるまで、プログラム内の後続のすべての処理ループに有効となります。

LIMIT ステートメントは、制限値が明示されている個々のステートメント（例：FIND(*n*) ...）には適用されません。

制限値に達すると処理が終了し、メッセージが表示されます。また、処理ループが制限値を超えた場合の反応を決定するセッションパラメータ LE も参照してください。

LIMIT ステートメントの指定がなければ、Natural インストール時に Natural プロファイルパラメータ LT と一緒に定義されたデフォルトのグローバル制限が使用されます。

レコードのカウント

処理ループが制限に達したかどうかを確定するために、ループ内で読み込んだ各レコードを制限に対してカウントします。 処理ループの制限には以下が適用されます。

- FIND や READ ステートメントの WHERE 節で除かれたレコードは、制限数の対象にはなりません。
- ACCEPT/REJECT ステートメントで排除されたレコードは、制限数にカウントされます。

構文説明

LIMIT *n*	制限値の指定 制限値 *n* は 0～4294967295（リーディングゼロはオプション）の数値定数で指定します。 制限値が "0" であれば、処理ループに入りません。

例

例 1 - LIMIT ステートメント

```
** Example 'LMTEX1': LIMIT
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
*
LIMIT 4
*
READ EMPLOY-VIEW BY NAME STARTING FROM 'BAKER'
  DISPLAY NOTITLE
          NAME PERSONNEL-ID CITY *COUNTER
END-READ
*
END
```

プログラム **LMTEX1** の出力：

```
        NAME         PERSONNEL         CITY            CNT
                        ID
-------------------- --------- -------------------- -----------

BAKER                20016700  OAK BROOK                      1
BAKER                30008042  DERBY                          2
```

```
BALBIN               60000110  BARCELONA                     3
BALL                 30021845  DERBY                         4
```

例 2 - LIMIT ステートメント（2 つのデータベースループに対して有効）

```
** Example 'LMTEX2': LIMIT (valid for two database loops)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
LIMIT 3
*
FIND EMPLOY-VIEW WITH NAME > 'A'
  READ EMPLOY-VIEW BY NAME STARTING FROM 'BAKER'
    DISPLAY NOTITLE 'CNT(0100)' *COUNTER(0100)
                    'CNT(0110)' *COUNTER(0110)
  END-READ
END-FIND
*
END
```

プログラム **LMTEX2** の出力：

```
 CNT(0100)   CNT(0110)
----------- -----------

          1           1
          1           2
          1           3
          2           1
          2           2
          2           3
          3           1
          3           2
          3           3
```

80 LOOP

[CLOSE] LOOP [(*r*)]

このchapterでは、次のトピックについて説明します。

機能

LOOP ステートメントは、処理ループを終了するために使用します。 ループの現在のパスの処理が終了し、制御が処理ループの先頭に戻ります。

LOOP ステートメントの発行対象の処理ループが終了した場合（つまり、すべてのレコードが処理されるか、反復が実行された場合）は、実行が LOOP ステートメントの後のステートメントから続行されます。

データベース変数参照

処理ループの終了に加えて、LOOP ステートメントは、ループ内に含まれる FIND、FIND FIRST、FIND UNIQUE、READ および GET ステートメントへのすべてのフィールド参照を除外します。

ビュー内のフィールドは、ビュー名を修飾子として使用して、処理ループの外部で参照できます。

制限事項

- 当ステートメントはレポーティングモードでだけ有効です。
- LOOP ステートメントは、IF や AT BREAK などの条件付きステートメントに基づいて指定することはできません。

構文説明

LOOP (*r*)	ステートメント参照表記： LOOP ステートメントは、ステートメントラベルまたは参照番号（表記 (*r*)）で指定できます。その場合、参照されているステートメントによって開始されたループまで（そのループを含む）のすべての内部ループが終了します。ステートメント参照が指定されていない場合は、最も内側のアクティブな処理ループが終了します。

例

例 **1**

```
0010 FIND ...
0020   READ ...
0030     READ ...
0040 LOOP (0010)    /* closes all loops
```

例 **2**

```
0010 FIND ...
0020   READ ...
0030     READ ...
0040     LOOP       /* closes loop initiated on line 0030
0050   LOOP         /* closes loop initiated on line 0020
0060 LOOP           /* closes loop initiated on line 0010
```

81 METHOD

```
METHOD method-name
 OF [INTERFACE] interface-name
 IS subprogram-name
END-METHOD
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CREATE OBJECT** | **DEFINE CLASS** | **INTERFACE** | **PROPERTY** | **SEND METHOD**

関連機能グループ：「*コンポーネントベースプログラミング*」

機能

METHOD ステートメントは、実装としてサブプログラムをメソッドにインターフェイス定義外で割り当てます。 問題のインターフェイス定義がコピーコードから取得され、クラス固有の方法で実装される場合は、これが使用されます。

METHOD ステートメントは、DEFINE CLASS ステートメント内でインターフェイス定義後にのみ使用できます。 指定されたインターフェイスとメソッドの名前は、DEFINE CLASS ステートメントの INTERFACE 節で定義されている必要があります。

構文説明

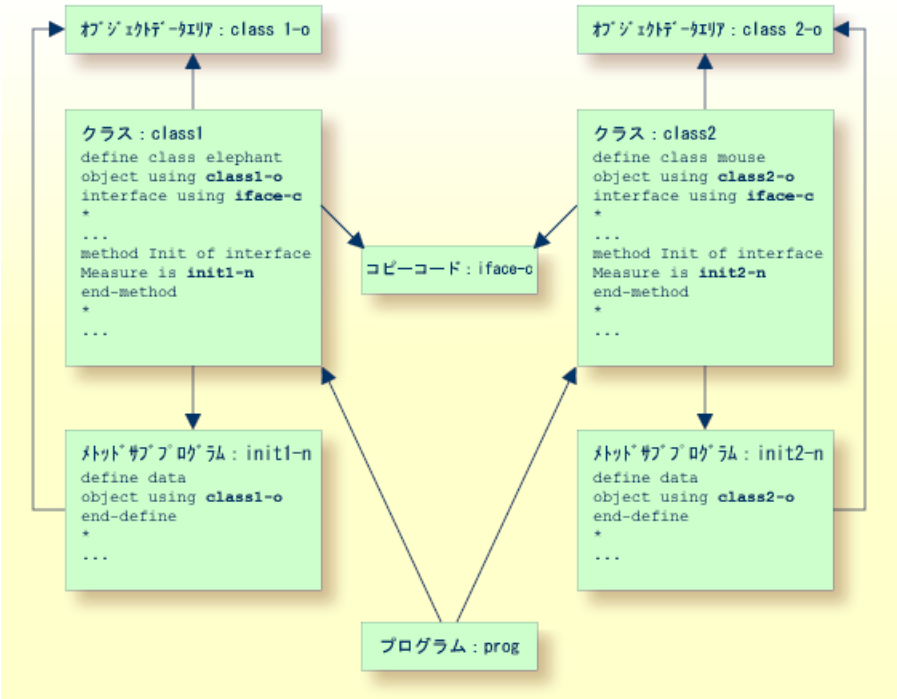
method-name	これは、メソッドに割り当てられた名前です。
OF *interface-name*	これは、インターフェイスに割り当てられた名前です。
IS *subprogram-name*	これは、メソッドを実装するサブプログラムの名前です。サブプログラムの名前は最大 8 文字です。 デフォルトは *method-name* です（IS 節が指定されない場合）。
END-METHOD	METHOD ステートメントを終了するには、Natural の予約語 END-METHOD を使用する必要があります。

例

以下の例は、同じインターフェイスを2つのクラスに別々に実装する方法と、これを実現するために PROPERTY ステートメントと METHOD ステートメントを使用する方法を示しています。

インターフェイス Measure をコピーコード iface-c に定義します。 クラス Elephant と Mouse はインターフェイス Measure を実装します。 したがって、それらは両方ともコピーコード iface-c を組み込みます。 しかし、クラスは、それらの個々のオブジェクトデータエリアから異なる変数を用いてプロパティ Height を実装し、異なるサブプログラムでメソッド Init を実装します。 それらは PROPERTY ステートメントを使用して選択したデータエリア変数をプロパティに割り当てて、METHOD ステートメントで選択したサブプログラムをメソッドに割り当てます。

プログラム prog は両方のクラスのオブジェクトを作成し、個々のクラス実装への初期化の指定をそのままにして、同じメソッド Init を使用してそれらを初期化できるようになります。



次は上記の例で使用された Natural モジュールの完全な内容を示しています。

コピーコード：**iface-c**

```
interface Measure
*
property Height(p5.2)
end-property
*
property Weight(i4)
end-property
*
method Init
end-method
*
end-interface
```

クラス：**class1**

```
define class elephant
object using class1-o
interface using iface-c
*
property Height of interface Measure is height
end-property
*
property Weight of interface Measure is weight
end-property
*
method Init of interface Measure is init1-n
end-method
*
end-class
end
```

オブジェクトデータエリア：**class1-o**

```
*   *** Top of Data Area ***
  1 HEIGHT                  P 5.2
  1 WEIGHT                  I 2
*   *** End of Data Area ***
```

メソッドサブプログラム：**init1-n**

```
define data
object using class1-o
end-define
*
height := 17.3
weight := 120
*
end
```

クラス：**class2**

```
define class mouse
object using class2-o
interface using iface-c
*
property Height of interface Measure is size
end-property
*
property Weight of interface Measure is weight
end-property
*
method Init of interface Measure is init2-n
end-method
*
end-class
end
```

オブジェクトデータエリア：**class2-o**

```
*   *** Top of Data Area ***
  1 SIZE                    P 3.2
  1 WEIGHT                  I 1
*   *** End of Data Area ***
```

メソッドサブプログラム：**init2-n**

```
define data
object using class2-o
end-define
*
size := 1.24
weight := 2
*
end
```

プログラム：**prog**

```
define data local
1 #o handle of object
1 #height(p5.2)
1 #weight(i4)
end-define
*
create object #o of class 'Elephant'
send "Init" to #o
#height := #o.Height
#weight := #o.Weight
write #height #weight
*
create object #o of class 'Mouse'
send "Init" to #o
#height := #o.Height
#weight := #o.Weight
write #height #weight
*
end
```

82 MOVE

MOVE ステートメントは、オペランドの値を 1 つ以上のオペランド（フィールドまたは配列）に移動するために使用します。

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ADD** | **COMPRESS** | **COMPUTE** | **DIVIDE** | **EXAMINE** | **MOVE ALL** | **MULTIPLY** | **RESET** | **SEPARATE** | **SUBTRACT**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

MOVE ステートメントは、オペランドの値を 1 つ以上のオペランド（フィールドまたは配列）に移動するために使用します。

複数のターゲットオペランドを持つ MOVE ステートメントは、対応する個々の MOVE ステートメントと同一です。

```
MOVE #SOURCE TO #TARGET1 #TARGET2
```

これは、以下と同じです。

```
MOVE #SOURCE TO #TARGET1
MOVE #SOURCE TO #TARGET2
```

例：

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:3) INIT <3,0,9>
1 #INDEX(I4)
1 #RESULT(I4)
END-DEFINE
*
#INDEX := 1
MOVE #ARRAY(#INDEX) TO #INDEX       /* #INDEX is 3
                       #RESULT      /* #RESULT is 9
*
#INDEX := 2
MOVE #ARRAY(#INDEX) TO  #INDEX     /* #INDEX is 0
                        #ARRAY(3)  /* returns run time error NAT1316
```

operand2 がダイナミック変数 変数のとき、その長さは MOVE 操作によって変更できます。 ダイナミック変数の現在の長さは、システム変数 *LENGTH を使用して確認できます。 ダイナミック

変数については、『*プログラミングガイド*』の「*ラージ変数／フィールドとダイナミック変数／フィールド*」を参照してください。

operand2 のフォーマットが C の場合、*operand1* は "(*parameter*)" として指定することもできます。 有効なパラメータは以下のとおりです。

MOVE ステートメントで指定可能なパラメータ		指定（S = ステートメントレベル、E = 要素レベル）
AD	属性定義	SE
CD	カラー定義	S

データ転送の互換性および規則についての詳細は、『*プログラミングガイド*』の「*データ転送*」を参照してください。

その他の考慮事項

受け取りフィールドにデータベースフィールドが指定されても、MOVE 処理はプログラム内のフィールドの内部値を更新するだけで、 データベースのフィールドの値は変更しません。

AT BREAK、AT END OF DATA または AT END OF PAGE ステートメントとともに MOVE ステートメントが指定された場合にのみ Natural システム関数が使用できます。

『*プログラミングガイド*』の「*演算割り当てのルール*」も参照してください。

Note: *operand1* が時間変数（フォーマット T）のとき、変数内容の時間コンポーネントだけが転送され、日付コンポーネントは転送されません（*構文 4* と *構文 5* で説明されている MOVE EDITED の場合を除く）。

構文説明

このステートメントには異なる構造が可能です。

- **構文 1 - MOVE ROUNDED**
- **構文 2 - MOVE SUBSTRING**
- **構文 3 - MOVE BY NAME / POSITION**
- **構文 4 - MOVE EDITED（*operand2* で指定された編集マスク）**
- **構文 5 - MOVE EDITED（*operand1* で指定された編集マスク）**
- **構文 6 - MOVE LEFT/RIGHT JUSTIFIED**
- **構文 7 - MOVE NORMALIZED**
- **構文 8 - MOVE ENCODED**

次の構文図で使用されている記号については、「*構文記号*」を参照してください。

構文 1 - MOVE ROUNDED

```
MOVE [ROUNDED] operand1 [( parameter)] TO operand2 ...
```

オペランド定義テーブル：

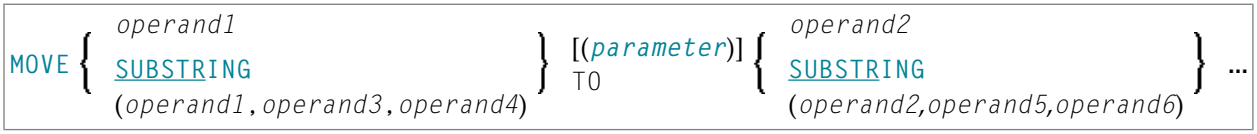
オペランド	構文要素					フォーマット												
operand1	C	S	A		N	A	U	N	P	I	F	B	D	T	L	C	G	O
operand2		S	A		M	A	U	N	P	I	F	B	D	T	L	C	G	O

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1	C	S	A		N	A	U	N	P	I	F	B	D	T	L	C	G	O	可	不可
operand2		S	A		M	A	U	N	P	I	F	B	D	T	L	C	G	O	可	可

構文要素の説明：

MOVE ROUNDED	当オプションは、*operand2* を四捨五入します。 ROUNDED が指定できるのは *operand2* が数字タイプのときだけです。 *operand2* のフォーマットが N または P で、*operand2* が複数回指定されている場合は、ROUNDED は、小数点後の 7 桁を持つターゲットオペランドについては無視されます。 「*例 1 - MOVE ステートメントのさまざまな使用例*」も参照してください。	
(*parameter*)	*parameter* として、オプション PM=I またはセッションパラメータ DF を指定できます。	
	PM=I	右から左に記述する言語をサポートするために、PM=I を指定して、*operand1* の値を逆（右から左）方向で *operand2* に渡すことができます。 例えば、次のステートメントの結果、#B の内容は "ZYX" となります。 `MOVE 'XYZ' TO #A` `MOVE #A (PM=I) TO #B` PM=I は、*operand2* が英数字フォーマットのときにのみ指定できます。 *operand1* のトレーリングブランクは削除され、その後の値が逆向きに並べ替えられ、*operand2* に移されます。*operand1* が英数字フォーマットではない場合には、英数字フォーマットに変換されてから、逆向きに並べられます。 PM=I と MOVE LEFT/RIGHT JUSTIFIED も参照してください。
	DF	*operand1* が日付変数で、*operand2* が英数字フィールドのとき、この日付変数に対するパラメータとして、セッションパラメータ DF を指定できます。セッションパラメータ DF については、『パラメータリファレンス』を参照してください。

構文 2 - MOVE SUBSTRING



オペランド定義テーブル：

オペランド	構文要素					フォーマット						ステートメント参照	ダイナミック定義
operand1	C	S	A			A	U				B	可	不可
operand2		S	A			A	U				B	可	不可
operand3	C	S						N	P	I	B*	可	不可
operand4	C	S						N	P	I	B*	可	不可
operand5	C	S						N	P	I	B*	可	不可
operand6	C	S						N	P	I	B*	可	不可

* 説明を参照

構文要素の説明：

MOVE SUBSTRING	SUBSTRING オプションの指定がないと、フィールドの内容全体が転送されます。 SUBSTRING オプションで、英数字、Unicode またはバイナリフィールドの一部分だけを転送できます。SUBSTRING 節のフィールド名（*operand1*）の後に、まず開始位置（*operand3*）、次に移動するフィールド部分の長さ（*operand4*）を指定します。 *operand1* の基礎となるフィールドフォーマットが以下に該当する場合： ■ 英数字（A）またはバイナリ（B）の場合は、*operand3* または *operand4* で提供される値はバイト数と見なされます。 ■ Unicode（U）の場合は、*operand3* または *operand4* で提供される値は Unicode コード単位数、つまりダブルバイトと見なされます。 例えば、フィールド #A の値の 5 桁目から 12 桁目をフィールド #B に転送するには、次のように指定します。 `MOVE SUBSTRING(#A,5,8) TO #B` *operand1* がダイナミック変数のとき、転送するフィールド部分は現在の長さ以下である必要があります。そうしないと、ランタイムエラーになります。 また、受け取りフィールドの一部分に英数字、Unicode またはバイナリを転送することもできます。SUBSTRING 節のフィールド名（*operand2*）の後に、まず開始位置（*operand5*）、次に値を移動するフィールド部分の長さ（*operand6*）を指定します。

	operand2 の基礎となるフィールドフォーマットが以下に該当する場合： ■ 英数字（A）またはバイナリ（B）の場合は、*operand5* または*operand6* で提供される値はバイト数と見なされます。 ■ Unicode（U）の場合は、*operand3* または*operand4* で提供される値は Unicode コード単位数、つまりダブルバイトと見なされます。 例えば、フィールド #A の値をフィールド #B の 3 桁目から 6 桁目に転送するには、次のように指定します。 `MOVE #A TO SUBSTRING(#B,3,4)` *operand2*がダイナミック変数のとき、その開始位置（*operand5*）は、現在の変数の長さに 1 を加えた長さ以下である必要があります。開始位置がその長さを超過すると、ランタイムエラーになります。これは、*operand2* の内容に確かでないギャップが生じるからです。 *operand3/5* または *operand4/6* がバイナリ変数の場合は、4 以下の長さでのみ使用できます。 *operand3/5* を省略すると、開始位置は "1" と見なされます。 *operand4/6* を省略すると、長さはフィールドの開始位置から終わりまでの範囲とみなされます。 *operand2*がダイナミック変数であり、その指定開始位置（*operand5*）が現在の変数の長さに 1 を加えた長さ（MOVE 処理が変数の長さを拡張するために使用されることを意味する）の場合、*operand6* は変数の新しい長さを決定するために指定する必要があります。 **注意:** SUBSTRING オプション指定の MOVE はバイトごとの転送です（『*プログラミングガイド*』の「*算術演算の規則*」に記述されている規則は適用されません）。

構文 3 - MOVE BY NAME / POSITION

オペランド定義テーブル：

オペランド	構文要素	フォーマット	ステートメント参照	ダイナミック定義
operand1	G		可	不可
operand2	G		可	不可

構文要素の説明：

MOVE BY NAME *operand1* **TO** *operand2*	当オプションは、あるデータ構造内の個々のフィールドを、その構造内の位置に関係なく別のデータ構造へと転送します。 ただし、フィールド名が両方の構造に存在する場合にのみ転送されます。これは、再定義で作成されたフィールドと同様に再定義されたフィールドも含みます。 個々のフィールドは、どのフォーマットでもかまいません。 オペランドにビューを指定することもできます。 **注意:** 個々の MOVE の順番は、*operand1* のフィールド順で決まります。 「*例 2 - MOVE BY NAME ステートメント*」も参照してください。 配列を伴う**MOVE BY NAME** データ構造が配列を持つ場合、MOVE時その配列に添字"(*)"が割り当てられます。その配列が割り当て処理の規則に合わない場合は、エラーを返します（『*プログラミングガイド*』の「*配列処理*」を参照）。 「*例 3 - 配列を伴う MOVE BY NAME*」も参照してください。
MOVE BY POSITION *operand1* **TO** *operand2*	当オプションは、フィールド名に関係なく、グループ内のフィールドの内容を別のグループに転送するために使用します。 値は定義されたフィールドの順にあるグループから他のグループへフィールドごとに転送されます（再定義で作成されたフィールドは入りません）。 個々のフィールドは、どのフォーマットでもかまいません。 各グループ内のフィールド数は同じにする必要があります。また、フィールドのレベル構造および配列の次元は一致している必要があります。フォーマット変換は、算術演算の規則に従って行われます。『*プログラミングガイド*』の「*演算割り当てのルール*」を参照してください。 オペランドにビューを指定することもできます。 「*例 4 - MOVE BY POSITION*」も参照してください。

構文 4 - MOVE EDITED（*operand2* で指定された編集マスク）

```
MOVE EDITED operand1 TO operand2  { (EM=value) }
                                  { (EMU=value) }
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1	C	S	A			A	U					B							可	不可
operand2		S	A			A	U	N	P	I	F	B	D	T	L				可	可

構文要素の説明：

MOVE EDITED	編集マスクが *operand2* に対して指定されている場合、*operand1* の値はこの編集マスクを使用して *operand2* に配置されます。 編集マスクは、*operand2* の入力編集マスクとみなすことができます。これは、*operand1* の英数字の内容のどの位置に *operand2* の大量の入力データが見つかるかを指定するために使用されます。 そのため、編集マスクが *operand2* に存在する以上の文字または桁を参照する場合は、編集マスクは切り捨てられます。 *operand1* の長さは、編集マスクで表される入力値の長さより短くすることはできません。 *operand1* が編集マスクの長さよりも長い場合は、余ったデータが無視されます。 *operand1* を編集マスクの長さより長くしないという前提条件のもとでは、次のステートメントを、 `MOVE EDITED operand1 TO operand2 (EM=value)` 次のステートメントの実行のような操作とみなすことができます。 `STACK TOP DATA operand1` `INPUT operand2 (EM=value)` 「*例 1 - MOVE ステートメントのさまざまな使用例*」も参照してください。
EM	編集マスクの詳細については、『パラメータリファレンス』でセッションパラメータ EM を参照してください。
EMU	Unicode 編集マスクの詳細については、『パラメータリファレンス』でセッションパラメータ EMU を参照してください。

構文 5 - MOVE EDITED（*operand1* で指定された編集マスク）

```
MOVE EDITED operand1 { (EM=value) } TO operand2
                     { (EMU=value)}
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S	A		N	A	U	N	P	I	F	B	D	T	L				可	不可
operand2		S	A			A	U					B							可	可

構文要素の説明：

MOVE EDITED	*operand1* に編集マスクが指定されると、指定の編集マスクを *operand1* に適用し、結果が *operand2* に転送されます。 編集マスクは、*operand1* の*出力*編集マスクとみなすことができます。これは、編集マスクで記述されたレイアウトおよび長さで英数字文字列を作成するために使用されます。 *operand1* のデータ文字または数字の他に、出力文字列に追加の装飾文字を含めることができます。 そのため、編集マスクが *operand1* に存在する以上の文字または桁を参照する場合は、編集マスクは切り捨てられます。作成された出力文字列（編集マスクが適用された後の *operand1* 値の結果）の長さが *operand2* の長さを超過しないようにする必要があります。 *operand2* を編集マスクの長さより短くしないという前提条件のもとでは、次のステートメントを `MOVE EDITED operand1 (EM=value) TO operand2` 次のステートメントのような操作とみなすことができます。 `WRITE operand1 (EM=value)` このステートメントは、画面に出力を表示せずに、変数 *operand2* に書き込みます。 「*例 1 - MOVE ステートメントのさまざまな使用例*」も参照してください。
EM	編集マスクの詳細については、『パラメータリファレンス』でセッションパラメータ EM を参照してください。
EMU	Unicode 編集マスクの詳細については、『パラメータリファレンス』でセッションパラメータ EMU を参照してください。

構文 6 - MOVE LEFT/RIGHT JUSTIFIED

```
MOVE { LEFT  } [JUSTIFIED] operand1 [(parameter)] TO operand2
     { RIGHT }
```

オペランド定義テーブル：

オペランド	構文要素			フォーマット	ステートメント参照	ダイナミック定義
operand1	C	S	A	N A U N P I F B D T L	可	不可
operand2		S	A	A U	可	可

構文要素の説明：

MOVE LEFT/RIGHT JUSTIFIED	当オプションを使用すると、値は *operand2* に左詰めまたは右詰めで転送されます。 *operand2*がダイナミック変数のとき、MOVE LEFT/RIGHT JUSTIFIEDは使用できません。
MOVE LEFT JUSTIFIED	MOVE LEFT JUSTIFIED で、*operand1* のリーディングブランクは削除されて、*operand2* に値が左詰めで置かれます。 *operand2* の残りは空白で埋められます。 値が *operand2* よりも長いときには、値の右側が切り捨てられます。
MOVE RIGHT JUSTIFIED	MOVE RIGHT JUSTIFIED で、*operand1* のトレーリングブランクは切り捨てられて、*operand2* に値が右詰めで置かれます。 *operand2* の残りは空白で埋められます。 値が *operand2* よりも長いときには、値の左側が切り捨てられます。 「*例 1 - MOVE ステートメントのさまざまな使用例*」も参照してください。
parameter	MOVE LEFT/RIGHT JUSTIFIED と PM=I を一緒に使用するとき、転送処理は次の手順で行われます。 1. *operand1* が英数字フォーマットではない場合、値は英数字フォーマットに変換されます。 2. *operand1* のトレーリングブランクが削除されます。 3. LEFT JUSTIFIED の場合、*operand1* のリーディングブランクも削除されます。 4. 値を逆に並べ替えてから *operand2* に移します。 5. 必要に応じて、*operand2*の残りを空白で充填するか、または値を切り捨てます（前述参照）。

構文 7 - MOVE NORMALIZED

MOVE NORMALIZED ステートメントは、Unicode 文字列を "Unicode Normalization Form C"（NFC）に変換します。 結果として生成される Unicode 文字列には、構成前の文字として使用可能な文字の結合順序が含まれません。

ターゲットオペランドの形式が Unicode でない場合は、Unicode からターゲットオペランドへの暗黙的な変換が行われます。この変換中に、デフォルトのコードページ（システム変数「*CODEPAGE」を参照）が使用されます。

MOVE NORMALIZED ステートメントの詳細は、『*Unicode およびコードページのサポート*』ドキュメントの「ステートメント」を参照してください。

構文図：

```
MOVE NORMALIZED operand1 TO operand2
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S	A				U										可	不可
operand2		S	A			A	U										可	可

構文要素の説明：

MOVE NORMALIZED	このオプションは、正規化されていない内容を含む可能性のある Unicode フィールドを "Unicode Normalization Form C"（NFC）に変換するために使用されます。 この複合形式の Unicode 文字列には、構成前の文字として使用可能な文字の結合順序が含まれません。 *http://www.unicode.org/reports/tr15/#Canonical_Composition_Examples*（"正規化フォーム D と C の例"）も参照してください。 例： `MOVE NORMALIZED #SCR TO #TGT`
operand1	変換する Unicode 文字列。
operand2	ターゲットオペランド。

例：

一部のコードポイントは、異なる Unicode 表現を持ちます。 例えば、ドイツ語の文字 'Ä' では、Unicode の分解表現は U+0041 に続いて U+0308 であり、結合文字（U+0308）を使用します。代

替表現は、構成前の文字 U+00C4 です。 MOVE NORMALIZED ステートメントは、可能な限り構成前の文字を使用して、結合文字を持つUnicode表現を正規化されたUnicode表現に変換します。

構文 8 - MOVE ENCODED

このセクションでは、MOVE ENCODED ステートメントの構文について説明します。 このステートメントの目的の詳細は、『*Unicode およびコードページのサポート*』ドキュメントの「ステートメント」を参照してください。

構文図：

```
MOVE ENCODED
  operand1 [[IN] CODEPAGE operand2] TO
  operand3 [[IN] CODEPAGE operand4]
  [GIVING operand5]
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S	A			A	U	B									可	不可
operand2		S				A	U										可	不可
operand3		S				A	U	B									可	可
operand4		S	A			A	U										可	不可
operand5		S							I4								可	可

構文要素の説明：

MOVE ENCODED	MOVE ENCODED ステートメントは、1 つのコードページにエンコードされている文字列を、別のコードページの同等の文字列に変換します。
operand1	変換される文字列。
operand2	*operand1* のコードページ。 *operand1* の形式が A または B の場合にのみ指定できます。注 1 を参照してください。
operand3	ターゲット。 変換結果がターゲットフィールドに適合しない場合は、結果がパディングまたは切り捨てられます。パディング文字には結果のコードページの空白が使用されます。 ダイナミック変数の長さは変換結果の長さに自動的に調整されるため、ターゲットフィールドがダイナミック変数として定義されている場合、パディングや切り捨ては不要です。
operand4	*operand3* のコードページ。 operand3 の形式が A または B の場合にのみ指定できます。 注 1 を参照してください。

operand5	キーワード GIVING がない場合は、エラーが発生したときに Natural エラーメッセージが返されます。 キーワード GIVING が使用されている場合、*operand5* では Natural エラーメッセージの代わりに 0 または Natural エラーコードが返されます。 ターゲットが切り捨てられた場合、Natural エラーメッセージは返されませんが、キーワード GIVING が使用されている場合は、*operand5* に切り捨てを示す適切なエラーコードが格納されます。

Notes:

1. コードページオペランドが指定されていない場合は、現在のセッションのコードページ（システム変数 *CODEPAGE の値）がデフォルトとして使用されます。
2. ステートメント SET GLOBALS またはシステムコマンド GLOBALS のセッションパラメータ CPCVERR が ON に設定されている場合は、ソースフィールドの少なくとも 1 文字を目的のコードページに正しく変換できなかった場合にエラーが出力されますが、ターゲットフィールドでは置換文字で置換されます。

例：

```
MOVE ENCODED A-FIELD1 TO A-FIELD2
```

無効：コードページ名がデフォルトで取得され、*operand1* と *operand3* に対して同じであるため、構文エラーになります。

```
MOVE ENCODED A-FIELD1 CODEPAGE 'IBM01140' TO A-FIELD2 CODEPAGE 'IBM01140'
```

無効：コーディングされたコードページ名が *operand1* と *operand3* に対して同じであるため、エラーになります。

```
MOVE ENCODED A-FIELD1 CODEPAGE 'IBM01140' TO A-FIELD2 CODEPAGE 'IBM037'
```

有効：IBM01140 でコーディングされている A-FIELD1 の文字列は、IBM037 でコーディングされた A-FIELD2 に変換されます。

```
MOVE ENCODED U-FIELD TO U-FIELD
```

無効：少なくとも 1 つのオペランドが A または B の形式である必要があるため、エラーになります。

```
MOVE ENCODED U-FIELD TO A-FIELD
```

有効：UTF-16 でエンコードされるものとみなされる U-FIELD の Unicode 文字列は、デフォルトのコードページ（*CODEPAGE）の英数字 A-FIELD に変換されます。

```
MOVE ENCODED A-FIELD TO U-FIELD
```

有効：デフォルトのコードページ（*CODEPAGE）でエンコードされるものとみなされる A-FIELD の文字列は、Unicode フィールド U-FIELD に変換されます。

```
MOVE ENCODED A100-FIELD CODEPAGE 'IBM1140' TO A50-FIELD CODEPAGE 'IBM037'
```

有効：関連するコードページを使用して、A100-FIELD（フォーマット／長さ：A100）から A50-FIELD（フォーマット／長さ：A50）に変換されます。 ターゲットは切り捨てられます。 Natural エラーメッセージは返されません。

```
MOVE ENCODED A100-FIELD CODEPAGE 'IBM1140' TO A50-FIELD CODEPAGE 'IBM037' GIVING
RC-FIELD
```

有効：関連するコードページを使用して、A100-FIELD（フォーマット／長さ：A100）から A50-FIELD（フォーマット／長さ：A50）に変換されます。 ターゲットは切り捨てられます。 GIVING 節が指定されていないため、RC-FIELD は、値の切り捨てが行われたことを示すエラーコードを受け取ります。

例

例 1 - MOVE ステートメントのさまざまな使用例

```
** Example 'MOVEX1': MOVE
************************************************************************
DEFINE DATA LOCAL
1 #A (N3)
1 #B (A5)
1 #C (A2)
1 #D (A7)
1 #E (N1.0)
1 #F (A5)
1 #G (N3.2)
1 #H (A6)
END-DEFINE
*
MOVE 5 TO #A
WRITE NOTITLE 'MOVE 5 TO #A'      30X '=' #A
*
MOVE 'ABCDE' TO #B #C #D
WRITE 'MOVE ABCDE TO #B #C #D'    20X '=' #B '=' #C '=' #D
*
MOVE -1  TO #E
WRITE 'MOVE -1  TO #E'            28X '=' #E
*
MOVE ROUNDED 1.995 TO #E
WRITE 'MOVE ROUNDED 1.995 TO #E'  18X '=' #E
*
*
MOVE RIGHT JUSTIFIED 'ABC' TO #F
WRITE 'MOVE RIGHT JUSTIFIED ''ABC'' TO #F'       10X '=' #F
*
MOVE EDITED '003.45' TO #G (EM=999.99)
WRITE 'MOVE EDITED ''003.45'' TO #G (EM=999.99)'  4X '=' #G
*
MOVE EDITED 123.45 (EM=999.99) TO #H
WRITE 'MOVE EDITED 123.45 (EM=999.99) TO #H'      6X '=' #H
*
END
```

プログラム **MOVEX1** の出力：

```
MOVE 5 TO #A                                #A:    5
MOVE ABCDE TO #B #C #D                      #B: ABCDE #C: AB #D: ABCDE
MOVE -1  TO #E                              #E: -1
MOVE ROUNDED 1.995 TO #E                    #E:  2
MOVE RIGHT JUSTIFIED 'ABC' TO #F            #F:   ABC
MOVE EDITED '003.45' TO #G (EM=999.99)      #G:    3.45
MOVE EDITED 123.45 (EM=999.99) TO #H        #H: 123.45
```

例 2 - MOVE BY NAME

```
** Example 'MOVEX2': MOVE BY NAME
************************************************************************
DEFINE DATA LOCAL
1 #SBLOCK
  2 #FIELDA (A10) INIT <'AAAAAAAAAA'>
  2 #FIELDB (A10) INIT <'BBBBBBBBBB'>
  2 #FIELDC (A10) INIT <'CCCCCCCCCC'>
  2 #FIELDD (A10) INIT <'DDDDDDDDDD'>
1 #TBLOCK
  2 #FIELD1 (A15) INIT <' '>
  2 #FIELDA (A10) INIT <' '>
  2 #FIELD2 (A10) INIT <' '>
  2 #FIELDB (A10) INIT <' '>
  2 #FIELD3 (A20) INIT <' '>
  2 #FIELDC (A10) INIT <' '>
END-DEFINE
*
MOVE BY NAME #SBLOCK TO #TBLOCK
*
WRITE NOTITLE 'CONTENTS OF #TBLOCK AFTER MOVE BY NAME:'
       // '=' #TBLOCK.#FIELD1
        / '=' #TBLOCK.#FIELDA
        / '=' #TBLOCK.#FIELD2
        / '=' #TBLOCK.#FIELDB
        / '=' #TBLOCK.#FIELD3
        / '=' #TBLOCK.#FIELDC
*
END
```

MOVE BY NAME 処理後の **#TBLOCK** の内容：

```
CONTENTS OF #TBLOCK AFTER MOVE BY NAME:

#FIELD1:
#FIELDA: AAAAAAAAAA
#FIELD2:
#FIELDB: BBBBBBBBBB
#FIELD3:
#FIELDC: CCCCCCCCCC
```

例 3 - 配列を伴う MOVE BY NAME

```
DEFINE DATA LOCAL
  1 #GROUP1
    2 #FIELD (A10/1:10)
  1 #GROUP2
    2 #FIELD (A10/1:10)
END-DEFINE
...
MOVE BY NAME #GROUP1 TO #GROUP2
...
```

この例では、MOVE ステートメントは内部的に次のように解決されます。

```
MOVE #GROUP1.#FIELD (*) TO #GROUP2.#FIELD (*)
```

添字付きグループの一部分が同一グループの他の部分に転送される場合、次の例のような予期しない結果となります。

```
DEFINE DATA LOCAL
  1 #GROUP1 (1:5)
    2 #FIELDA (N1) INIT <1,2,3,4,5>
    2 REDEFINE #FIELDA
      3 #FIELDB (N1)
END-DEFINE
...
```

```
MOVE BY NAME #GROUP1 (2:4) TO #GROUP1 (1:3)
...
```

この例では、MOVE ステートメントは内部的に次のように解決されます。

```
MOVE #FIELDA (2:4) TO #FIELDA (1:3)
MOVE #FIELDB (2:4) TO #FIELDB (1:3)
```

まず、#FIELDA のオカレンス 2 から 4 の内容は、#FIELDA のオカレンス 1 から 3 に移動されます。つまり、オカレンスは次の値を受け取ります。

オカレンス	1.	2.	3.	4.	5.
前の値：	1	**2**	**3**	**4**	5
後の値：	**2**	**3**	**4**	4	5

#FIELDB のオカレンス 2 から 4 の内容は、#FIELDB のオカレンス 1 から 3 に移動されます。つまり、オカレンスは次の値を受け取ります。

オカレンス	1.	2.	3.	4.	5.
前の値：	2	**3**	**4**	**4**	5
後の値：	**3**	**4**	**4**	4	5

例 4 - MOVE BY POSITION

```

DEFINE DATA LOCAL
  1 #GROUP1
    2 #FIELD1A (N5)
    2 #FIELD1B (A3/1:3)
    2 REDEFINE #FIELD1B
      3 #FIELD1BR (A9)
  1 #GROUP2
    2 #FIELD2A (N5)
    2 #FIELD2B (A3/1:3)
    2 REDEFINE #FIELD2B
      3 #FIELD2BR (A9)
END-DEFINE
...
MOVE BY POSITION #GROUP1 TO #GROUP2
...
```

#FIELD1A の内容は、#FIELD2A に転送され、#FIELD1B の内容は、#FIELD2B に転送されます。フィールド #FIELD1BR と #FIELD2BR は影響を受けません。

83 MOVE ALL

```
MOVE ALL operand1 TO operand2 [UNTIL operand3]
```

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ADD** | **COMPRESS** | **COMPUTE** | **DIVIDE** | **EXAMINE** | **MOVE** | **MULTIPLY** | **RESET** | **SEPARATE** | **SUBTRACT**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

MOVE ALL ステートメントは、*operand1* の値を *operand3* がいっぱいになるまで *operand2* に繰り返し移動するために使用します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A	U	N				B						可	不可
operand2		S	A			A	U					B						可	可
operand3	C	S						N	P	I								可	不可

構文要素の説明：

operand1	ソースオペランド： ソースオペランドには転送する値が含まれています。 リーディングゼロを含む数値オペランドのすべての桁が転送されます。
TO *operand2*	ターゲットオペランド： ターゲットオペランドは、MOVE ALL 処理の前にリセットされません。 このことは UNTIL オプションを使用するときに特に重要です。*operand2* にあらかじめ存在していたデータは、MOVE ALL 処理で明確に上書きしないと、そのまま残されるからです。
UNTIL *operand3*	**UNTIL** オプション： UNTIL オプションは *operand2* に対する桁数を指定することにより MOVE ALL 処理を制限するために使用します。 *Operand3* には桁数を指定します。 MOVE ALL 処理はこの値に達したときに終了します。

	operand3 が *operand2* の長さより大きい場合、MOVE ALL 処理は *operand2* がいっぱいになったときに終了します。 UNTIL オプションを使用して、ダイナミック変数に初期値を割り当てることもできます。*operand2* がダイナミック変数のとき、MOVE ALL 操作後のダイナミック変数の長さは、*operand3* の値に対応します。ダイナミック変数の現在の長さは、システム変数 *LENGTH を使用して確認できます。ダイナミック変数については、「*ダイナミック変数の使用*」を参照してください。

例

```
** Example 'MOAEX1': MOVE ALL
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 4
RD. READ EMPLOY-VIEW BY NAME
  SUSPEND IDENTICAL SUPPRESS
  /*
  FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      MOVE ALL '*' TO FIRST-NAME (RD.)
      MOVE ALL '*' TO CITY (RD.)
      MOVE ALL '*' TO MAKE (FD.)
    END-NOREC
    /*
    DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
            NAME (RD.) FIRST-NAME (RD.)
            CITY (RD.)
            MAKE (FD.) (IS=OFF)
    /*
  END-FIND
END-READ
END
```

プログラム **MOAEX1** の出力：

```
     NAME          FIRST-NAME          CITY             MAKE
--------------- --------------- --------------- ---------------

ABELLAN         *************** *************** ***************
ACHIESON        ROBERT          DERBY           FORD
ADAM            *************** *************** ***************
ADKINSON        JEFF            BROOKLYN        GENERAL MOTORS
```

84 MOVE INDEXED

MOVE INDEXED ステートメントは、互換性保持のためにのみサポートされています。

Caution: 配列オペランドを伴う MOVE ステートメントとは対照的に、MOVE INDEXED ステートメントが実行される場合に範囲外の添字値はチェックできません。 そのため、不正な MOVE INDEXED ステートメントを実行すると、ユーザーデータが不用意に破壊されることがあります。

したがって、MOVE INDEXED ステートメントを MOVE ステートメントで置き換えることを強くお勧めします。

詳細については、「MOVE」ステートメントを参照してください。

85 MULTIPLY

MULTIPLY ステートメントは、2 つのオペランドを乗算するために使用します。

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ADD** | **COMPRESS** | **COMPUTE** | **DIVIDE** | **EXAMINE** | **MOVE** | **MOVE ALL** | **RESET** | **SEPARATE** | **SUBTRACT**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

MULTIPLY ステートメントは、2 つのオペランドを乗算するために使用します。 使用される構文に応じて、乗算結果は *operand1* または *operand3* に入ります。

データベースフィールドを結果フィールドとして使用するとき、MULTIPLY 演算は、プログラム内で使用したフィールドの内部値だけを更新します。 データベース内のフィールドの値は変更されないまま維持されます。

配列を含む乗算については、『*プログラミングガイド*』の「*算術演算の規則*」、「*配列での算術演算*」も参照してください。

構文説明

このステートメントには、2 つの異なる構造が可能です。

- **構文** 1 - GIVING **節のない** MULTIPLY
- **構文** 2 - GIVING **節のある** MULTIPLY

構文 1 - GIVING 節のない MULTIPLY

構文 1 が使用される場合、乗算の結果は *operand1* に入ります。

```
MULTIPLY [ROUNDED] operand1 BY operand2
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

オペランド定義テーブル（構文 1）：

オペランド	構文要素					フォーマット										ステートメント参照	ダイナミック定義
operand1		S	A		M		N	P	I	F						可	不可
operand2	C	S	A		N		N	P	I	F						可	不可

構文要素の説明（構文 1）

operand1 **BY** *operand2*	*operand1* は被乗数、*operand2* は乗数です。GIVING 節が使用されていない場合は、*operand1* に結果が入るため、ステートメントは次と同等になります。 `<oper1> := <oper1> * <oper2>`
ROUNDED	キーワード ROUNDED を指定すると、値は *operand1* または *operand3* に割り当てられる前に切り上げられます。切り上げについては、『*プログラミングガイド*』の「*演算割り当てのルール*」、「*フィールドの切り捨てと切り上げ*」を参照してください。

構文 2 - GIVING 節のある MULTIPLY

構文 2 が使用される場合、乗算の結果は *operand3* に入ります。

```
MULTIPLY [ROUNDED] operand1 BY operand2 GIVING operand3
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

オペランド定義テーブル（構文 2）：

オペランド	構文要素					フォーマット														ステートメント参照	ダイナミック定義
operand1	C	S	A		M			N	P	I	F									可	不可
operand2	C	S	A		N			N	P	I	F									可	不可
operand3		S	A		M	A	U	N	P	I	F	B*		T						可	可

**operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明（構文 2）

operand1 **BY** *operand2* **GIVING** *operand3*	*operand1* は被乗数、*operand2* は乗数です。GIVING 節が使用されている場合は、*operand1* が変更されず、結果が *operand3* に入るため、ステートメントは次と同等になります。 `<oper3> := <oper1> * <oper2>` *operand1* が数値定数の場合は、GIVING 節が必要です。
ROUNDED	キーワード ROUNDED を指定すると、値は *operand1* または *operand3* に割り当てられる前に切り上げられます。切り上げについては、『*プログラミングガイド*』の「*演算割り当てのルール*」、「*フィールドの切り捨てと切り上げ*」を参照してください。

例

```
** Example 'MULEX1': MULTIPLY
************************************************************************
DEFINE DATA LOCAL
1 #A      (N3) INIT <20>
1 #B      (N5)
1 #C      (N3.1)
1 #D      (N2)
1 #ARRAY1 (N5/1:4,1:4) INIT (2,*) <5>
1 #ARRAY2 (N5/1:4,1:4) INIT (4,*) <10>
END-DEFINE
*
MULTIPLY #A BY 3
WRITE NOTITLE 'MULTIPLY #A BY 3'            25X '=' #A
*
MULTIPLY #A BY 3 GIVING #B
WRITE 'MULTIPLY #A BY 3 GIVING #B'          15X '=' #B
*
MULTIPLY ROUNDED 3 BY 3.5 GIVING #C
WRITE 'MULTIPLY ROUNDED 3 BY 3.5 GIVING #C'  6X '=' #C
*
MULTIPLY 3 BY -4 GIVING #D
WRITE 'MULTIPLY  3 BY -4 GIVING #D'         14X '=' #D
*
MULTIPLY -3 BY -4 GIVING #D
WRITE 'MULTIPLY -3 BY -4 GIVING #D'         14X '=' #D
*
MULTIPLY  3 BY  0 GIVING #D
WRITE 'MULTIPLY  3 BY  0 GIVING #D'         14X '=' #D
*
```

```
WRITE / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)
WRITE / 'MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)'
      / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
*
END
```

プログラム **MULEX1** の出力：

```
MULTIPLY #A BY 3                          #A:   60
MULTIPLY #A BY 3 GIVING #B                #B:    180
MULTIPLY ROUNDED 3 BY 3.5 GIVING #C       #C:   10.5
MULTIPLY  3 BY -4 GIVING #D               #D: -12
MULTIPLY -3 BY -4 GIVING #D               #D:  12
MULTIPLY  3 BY  0 GIVING #D               #D:   0

#ARRAY1:      5      5      5      5 #ARRAY2:     10     10     10     10

MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)
#ARRAY1:     50     50     50     50 #ARRAY2:     10     10     10     10
```

86 NEWPAGE

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

NEWPAGE ステートメントは、ページ送りをするために使用します。 また、NEWPAGE では AT END OF PAGE および WRITE TRAILER ステートメントも実行されます。 WRITE TITLE、WRITE NOTITLE あるいは DISPLAY NOTITLE ステートメントを指定して固有のタイトル処理を定義していない限り、日付、時刻、ページ番号を含むデフォルトのタイトルが新しいページごとに出力されます。

Notes:

1. ページ送りは、NEWPAGE ステートメントが実行されるときに行われるわけではありません。出力を生成する後続のステートメントが実行されるときにのみ行われます。
2. NEWPAGE ステートメントを使用しないとき、ページ送りは Natural セッションパラメータ PS に基づいて自動的に制御されます。

構文説明

オペランド定義テーブル：

オペランド	構文要素						フォーマット										ステートメント参照	ダイナミック定義
operand1	C	S					N	P	I								可	不可

構文要素の説明：

(*rep***)**	レポート指定： 表記（*rep*）は、NEWPAGE ステートメントを適用するレポートの ID を指定するために使用できます。 範囲 0～31 の値、または **DEFINE PRINTER** ステートメントを使用して割り当てた論理名を指定できます。 （*rep*）を指定しない場合、NEWPAGE ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。
EVEN IF TOP OF PAGE	当オプションは、NEWPAGE ステートメントの実行の直前にページ送りが行われていても、新しいページを（対応する AT TOP OF PAGE およびページタイトル処理とともに）生成します。
WHEN LESS THAN *operand1* **LINES LEFT**	当オプションは、現ページの残りの行数が *operand1* で指定された行数よりも少ないとき、ページ換えをするために使用します（現在の行はセッションパラメータ PS の値と比較されます）。
WITH TITLE	WITH TITLE オプションは、新しいページに変わったときのタイトルを指定するために使用します。 タイトルの指定は、NEWPAGE WITH TITLE ステートメントでは SKIP 節が使用できないことを除けば、WRITE TITLE ステートメントの構文と同じです。

例

```
** Example 'NWPEX1': NEWPAGE
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 15
READ EMPLOY-VIEW BY CITY FROM 'DENVER'
  DISPLAY CITY (IS=ON) NAME SALARY (1) CURR-CODE (1)
  AT BREAK OF CITY
    SKIP 1
    /*
    NEWPAGE WHEN LESS THAN 10 LINES LEFT
    WRITE '****************************************'
      /   'SUMMARY FOR ' OLD(CITY)
      /   '****************************************'
      /   '****************************************'
      /   'SUM OF SALARIES:' SUM(SALARY(1))
      /   'AVG OF SALARIES:' AVER(SALARY(1))
      /   '****************************************'
    NEWPAGE
    /*
  END-BREAK
END-READ
END
```

プログラム **NWPEX1** の出力 **-** ページ **1**：

```
Page      1                                                 05-01-18  10:01:45

        CITY                 NAME           ANNUAL   CURRENCY
                                            SALARY     CODE
------------------- -------------------- ---------- --------

DENVER              TANIMOTO                  33000 USD
                    MEYER                     50000 USD

****************************************
SUMMARY FOR  DENVER
****************************************
****************************************
```

```
SUM OF SALARIES:      83000
AVG OF SALARIES:      41500
****************************************
```

プログラム **NWPEX1** の出力 **-** ページ **2**：

```
Page      2                                                  05-01-18  10:01:45

        CITY                 NAME           ANNUAL   CURRENCY
                                            SALARY     CODE
------------------- -------------------- ---------- --------

DERBY               DEAKIN                     8750 UKL
                    GARFIELD                   6750 UKL
                    MUNN                       8800 UKL
                    MUNN                       5650 UKL
                    GREBBY                     9550 UKL
                    WHITT                      8650 UKL
                    PONSONBY                   5500 UKL
                    MAGUIRE                    4150 UKL
                    HEYWOOD                    3900 UKL
                    BRYDEN                     6750 UKL
                    SMITH                     39000 UKL
                    CONQUEST                  45000 UKL
                    ACHIESON                  11300 UKL

****************************************
SUMMARY FOR  DERBY
****************************************
```

プログラム **NWPEX1** の出力 **-** ページ **3**：

```

------------------- -------------------- ---------- --------

DERBY               DEAKIN                     8750 UKL
                    GARFIELD                   6750 UKL
                    MUNN                       8800 UKL
                    MUNN                       5650 UKL
                    GREBBY                     9550 UKL
                    WHITT                      8650 UKL
                    PONSONBY                   5500 UKL
                    MAGUIRE                    4150 UKL
                    HEYWOOD                    3900 UKL
                    BRYDEN                     6750 UKL
                    SMITH                     39000 UKL
                    CONQUEST                  45000 UKL
                    ACHIESON                  11300 UKL

****************************************
```

```
SUMMARY FOR  DERBY
**************************************
**************************************
SUM OF SALARIES:     163750
AVG OF SALARIES:      12596
**************************************
```

87 OBTAIN

```
OBTAIN operand1 ...
```

このchapterでは、次のトピックについて説明します。

機能

OBTAIN ステートメントは、1 つ以上のフィールドをファイルから読み取るときに使用します。OBTAIN ステートメントは、Natural オブジェクトプログラムの実行コードを生成するものではありません。 主に、マルチプルバリューフィールドの値範囲またはピリオディックグループのオカレンスの範囲を読み取って、これらの範囲の一部を後でプログラムで参照できるようにするために使用されます。

Natural は後続のステートメント（DISPLAY ステートメントや COMPUTE ステートメントなど）で参照される各データベースフィールドを自動的に読み取るため、プログラムで参照するデータベースフィールドごとに OBTAIN ステートメントを使用する*必要はありません*。

配列形式のマルチプルバリューフィールドまたはピリオディックグループフィールドが参照されている場合は、配列を OBTAIN ステートメントで定義して、フィールドのすべてのオカレンスに対して作成されるようにする必要があります。配列が定義される前に個々のマルチプルバリューフィールドまたはピリオディックグループフィールドが参照された場合は、フィールドが配列に配置されず、配列からは独立して存在するようになります。 フィールドには、配列内の対応するオカレンスと同じ値が含まれます。

2つ目の個々のオカレンスまたは配列の配列次元が初期配列に含まれている場合、マルチプルバリューフィールド、ピリオディックグループフィールドまたはサブ配列の個々のオカレンスは、以前に定義した配列に保持できます。

一意の変数添字のあるマルチプルバリューフィールドまたはピリオディックグループフィールドへの参照は、値の配列に含めることができません。 配列の個々のオカレンスを変数添字とともに処理する場合は、添字表現の接頭辞として一意の変数添字を付けて、個々の配列を示す必要があります。

制限事項

OBTAIN ステートメントはレポーティングモードでだけ有効です。

構文説明

オペランド定義テーブル：

オペランド	構文要素				フォーマット													ステートメント参照	ダイナミック定義
operand1		S	A	G		A	U	N	P	I	F	B	D	T	L				可	可

構文要素の説明：

operand1	*operand1* で、OBTAIN ステートメントの結果、参照可能になるフィールドを指定します。

例：

```
READ FINANCE OBTAIN CREDIT-CARD (1-10)
DISPLAY CREDIT-CARD (3-5) CREDIT-CARD (6-8)
SKIP 1 END
```

上記の例では、フィールド CREDIT-CARD（ピリオディックグループに含まれます）の最初の 10 個のオカレンスが読み取られ、オカレンス3～5と6～8が表示されます。後続のサブ配列は初期配列（1～10）に存在するようになります。

```
READ FINANCE
MOVE 'ONE' TO CREDIT-CARD (1)
DISPLAY CREDIT-CARD (1) CREDIT-CARD (1-5)
```

出力：

```
    CREDIT-CARD      CREDIT-CARD
------------------ ----------------


ONE                DINERS CLUB
                   AMERICAN EXPRESS




ONE                AVIS
                   AMERICAN EXPRESS
```

```
ONE                HERTZ
                   AMERICAN EXPRESS



ONE                UNITED AIR TRAVEL
```

CREDIT-CARD (1)への最初の参照は、配列に含まれていません。一意のオカレンス（1）への参照の後に定義されている配列は、一意のオカレンスまたは定義されている配列より短い配列を遡及的に含むことはできません。

```
READ FINANCE
OBTAIN CREDIT-CARD (1-5)
MOVE 'ONE' TO CREDIT-CARD (1)
DISPLAY CREDIT-CARD (1) CREDIT-CARD (1-5)
```

出力：

```
    CREDIT-CARD      CREDIT-CARD
------------------ ----------------


ONE                ONE
                   AMERICAN EXPRESS



ONE                ONE
                   AMERICAN EXPRESS



ONE                ONE
                   AMERICAN EXPRESS
```

```
ONE                ONE
```

CREDIT-CARD (1) への個々の参照は、OBTAIN ステートメントで定義されている配列に含まれます。

```
MOVE (1) TO INDEX
READ FINANCE
DISPLAY CREDIT-CARD (1-5) CREDIT-CARD (INDEX)
```

出力：

```
    CREDIT-CARD      CREDIT-CARD
------------------- ----------------


DINERS CLUB         DINERS CLUB
AMERICAN EXPRESS



AVIS                AVIS
AMERICAN EXPRESS



HERTZ               HERTZ
AMERICAN EXPRESS



UNITED AIR TRAVEL  UNITED AIR TRAVEL
```

変数添字表記を使用した CREDIT-CARD への参照は、配列に含まれません。

```
RESET A(A20) B(A20) C(A20)
MOVE 2 TO I (N3)
MOVE 3 TO J (N3)
READ FINANCE
 OBTAIN CREDIT-CARD (1:3) CREDIT-CARD (I:I+2) CREDIT-CARD (J:J+2)
 FOR K (N3) = 1 TO 3
  MOVE CREDIT-CARD (1.K) TO A
  MOVE CREDIT-CARD (I.K) TO B
  MOVE CREDIT-CARD (J.K) TO C
  DISPLAY A B C
 LOOP /* FOR
```

```
LOOP / * READ
END
```

出力：

```
         A                   B                   C
------------------- ------------------- -------------------
CARD 01             CARD 02             CARD 03
CARD 02             CARD 03             CARD 04
CARD 03             CARD 04             CARD 05
```

3つの配列は、一意の基本添字を添字表現の修飾子として使用して、個別にアクセスできます。

無効な例 **1**

```
READ FINANCE
OBTAIN CREDIT-CARD (1-10)
FOR I 1 10
MOVE CREDIT-CARD (I) TO A(A20)
WRITE A
END
```

上記の例では、レコードの読み取り時（READ）に添字 I にまだ値 0 が含まれているため、エラーメッセージ NAT1006（変数添字の値 = 0）が生成されます。

いずれの場合にも、上記の例では、変数添字を持つ個々のオカレンスを配列に含めることができず、変数添字（I）は次のレコードの読み取り時にのみ評価されるため、CREDIT-CARD の最初の 10 個のオカレンスは出力されません。

上記を実行する正しいメソッドは次のとおりです。

```
READ FINANCE
OBTAIN CREDIT-CARD (1-10)
FOR I 1 10
MOVE CREDIT-CARD (1.I) TO A (A20)
WRITE A
END
```

無効な例 **2**

```
READ FINANCE
FOR I 1 10
WRITE CREDIT-CARD (I)
END
```

上記の例では、レコードが READ ステートメントで読み取られたときに添字 I がゼロであるため、エラー メッセージ NAT1006 が生成されます。

上記を実行する正しいメソッドは次のとおりです。

```
READ FINANCE
FOR I 1 10
GET SAME
WRITE CREDIT-CARD (0030/I)
END
```

GET SAME ステートメントは、変数添字が FOR ループ内で更新された後にレコードを再び読み取るために必要です。

例

例 1 - OBTAIN ステートメント

```
** Example 'OBTEX1': OBTAIN
************************************************************************
RESET #INDEX (I1)
*
LIMIT 5
READ EMPLOYEES BY CITY
  OBTAIN SALARY (1:4)
  /*
  IF SALARY (4) GT 0 DO
    WRITE '=' NAME / 'SALARIES (1:4):' SALARY (1:4)
    FOR #INDEX 1 TO 4
      WRITE 'SALARY' #INDEX  SALARY (1.#INDEX)
    LOOP
    SKIP 1
  DOEND
LOOP
```

```
*
END
```

プログラム **OBTEX1** の出力：

```
Page      1                                                  05-02-08  13:37:48

NAME: SENKO
SALARIES (1:4):      31500      29900      28100      26600
SALARY    1      31500
SALARY    2      29900
SALARY    3      28100
SALARY    4      26600

NAME: HAMMOND
SALARIES (1:4):      22000      20200      18700      17500
SALARY    1      22000
SALARY    2      20200
SALARY    3      18700
SALARY    4      17500
```

例 2 - 複数の範囲がある OBTAIN ステートメント

```
** Example 'OBTEX2': OBTAIN (with multiple ranges)
************************************************************************
RESET #INDEX (I1) #K (I1)
*
#INDEX  := 2
#K      := 3
*
LIMIT 2
*
READ EMPLOYEES BY CITY
  OBTAIN SALARY (1:5)
         SALARY (#INDEX:#INDEX+3)
  /*
  IF SALARY (5) GT 0 DO
    WRITE '=' NAME
    WRITE 'SALARIES (1-5):' SALARY (1:5) /
    WRITE 'SALARIES (2-5):' SALARY (#INDEX:#INDEX+3)
    WRITE 'SALARIES (2-5):' SALARY (#INDEX.1:4) /
    WRITE 'SALARY 3:' SALARY (3)
    WRITE 'SALARY 3:' SALARY (#K)
    WRITE 'SALARY 4:' SALARY (#INDEX.#K)
  DOEND
LOOP
```

プログラム **OBTEX2** の出力：

```
Page      1                                                05-02-08  13:38:31

NAME: SENKO
SALARIES (1-5):      31500      29900      28100      26600      25200

SALARIES (2-5):      29900      28100      26600      25200
SALARIES (2-5):      29900      28100      26600      25200

SALARY 3:      28100
SALARY 3:      28100
SALARY 4:      26600
```

OBTAIN ステートメントの詳細な使用例については、を参照してください。

「*データベース配列の参照*」（『*プログラミングガイド*』）

88 ON ERROR

ストラクチャードモード構文

```
ON ERROR
 statement ...
END-ERROR
```

レポーティングモード構文

```
ON ERROR  { statement ...          }
          { DO statement ... DOEND }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DECIDE FOR** | **DECIDE ON** | **IF** | **IF SELECTION**

関連機能グループ：「*論理条件の処理*」

機能

ON ERROR ステートメントは実行時のエラーを拾い、Natural エラーメッセージを別の形で出力し、Natural プログラムの実行を終了させ、コマンド入力モードに戻すために使用します。

ON ERROR ステートメントブロックの実行が開始されると、プログラムの正常な流れは中断され、再開することはできません。ただしエラー 3145（要求したレコードがホールド状態にある）の場合は、RETRY ステートメントによって処理が中断された所から再開されます。

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

制限事項

各 Natural オブジェクト内で 1 回だけ、ON ERROR ステートメントを使用できます。

構文説明

statement...	**ON ERROR** 処理の定義： ON ERROR 条件が発生したときに実行される処理を定義するには、1 つまたは複数のステートメントを指定できます。 **ON ERROR** ブロックからの抜け出し FETCH、STOP、TERMINATE、RETRY または ESCAPE ROUTINE の各ステートメントを使用して、ON ERROR ブロックから抜け出すことができます。 これらのステートメントのいずれかを使用して抜け出さないと、通常のエラーメッセージ処理が行われ、プログラムの実行が終了します。
END-ERROR	Natural 予約語 END-ERROR を使用して、ON ERROR ステートメントブロックを終了させる必要があります。

サブルーチン内の ON ERROR 処理

CALLNAT、PERFORM または FETCH RETURN を使用したサブルーチン構成になっている場合、各モジュールに ON ERROR ステートメントが指定可能です。

エラーが起きた場合、Naturalは自動的にサブルーチン構成に従って戻り、実行中のサブルーチン内で最初に見つかった ON ERROR ステートメントを選択します。 ON ERROR ステートメントがどのレベルのどのモジュールにもない場合は、通常のエラーメッセージ処理を行い、プログラムの実行が終了します。

システム変数 *ERROR-NR および *ERROR-LINE

次の Natural システム変数は、（次の例で示すように）ON ERROR ステートメントと組み合わせて使用できます。

***ERROR-NR**	Natural で検出されたエラーの数が含まれます。
***ERROR-LINE**	エラーの原因となったステートメントの行番号が含まれます。

例

```
** Example 'ONEEX1': ON ERROR
**
**
CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
*
1 #NAME (A20)
1 #CITY (A20)
END-DEFINE
*
REPEAT
  INPUT 'ENTER NAME:' #NAME
  IF #NAME = ' '
    STOP
  END-IF
  FIND EMPLOY-VIEW WITH NAME = #NAME
    INPUT (AD=M) 'ENTER NEW VALUES:' ///
                 'NAME:' NAME /
                 'CITY:' CITY
    UPDATE
    END TRANSACTION
    /*
    ON ERROR
      IF *ERROR-NR = 3009
        WRITE 'LAST TRANSACTION NOT SUCCESSFUL'
            / 'HIT ENTER TO RESTART PROGRAM'
        FETCH 'ONEEX1'
      END-IF
      WRITE 'ERROR' *ERROR-NR 'OCCURRED IN PROGRAM' *PROGRAM
            'AT LINE' *ERROR-LINE
      FETCH 'MENU'
    END-ERROR
    /*
  END-FIND
END-REPEAT
END
```

89 OPEN CONVERSATION

```
OPEN CONVERSATION USING [SUBPROGRAMS] {operand1} ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CLOSE CONVERSATION** | **DEFINE DATA CONTEXT**

機能

`OPEN CONVERSATION` ステートメントは、Natural リモートプロシージャコール（RPC）と一緒に使用します。 これにより、クライアントは会話を開き、会話に取り込むリモートサブプログラムを指定することができます。

`OPEN CONVERSATION` ステートメントを実行すると、会話を識別するユニーク ID がシステム変数 `*CONVID` に割り当てられます。

構文説明

オペランド定義テーブル：

オペランド	構文要素	フォーマット	ステートメント参照	ダイナミック定義
operand1	C S A	A	可	不可

構文要素の説明：

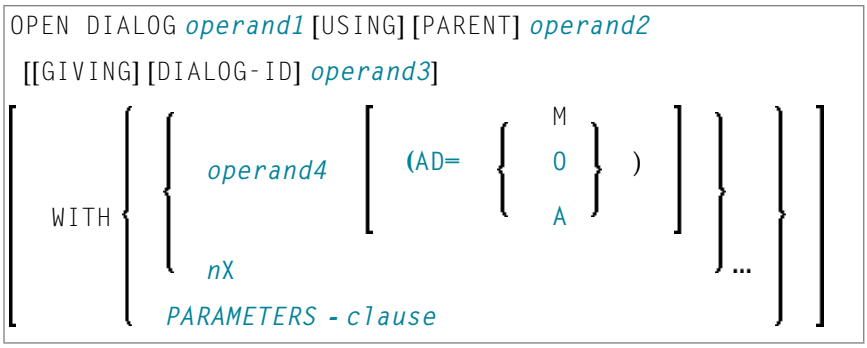
operand1	サブプログラム名： *operand1* として、会話に取り込むリモートサブプログラムの名前を指定します。 サブプログラム名は 1～8 文字の定数または 1～8 桁の英数字変数で指定することができます。

詳細と例

『*Natural リモートプロシージャコール（RPC）*』ドキュメントの次のセクションを参照してください。

- *会話型モードでの Natural RPC の動作*
- *会話型 RPC の使用*

90 OPEN DIALOG



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CLOSE DIALOG** | **PROCESS GUI** | **SEND EVENT**

関連機能グループ：「*イベントドリブンプログラミング*」

機能

このステートメントは、ダイナミックにダイアログを開きます。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A												可	不可
operand2	C	S														G		不可	不可
operand3		S								I								可	不可
operand4	C	S	A			A	U	N	P	I	F	B	D	T	L	C	G	O	可	不可

構文要素の説明：

operand1	ダイアログ名： *operand1* は、開かれるダイアログボックスの名前です。 *PARAMETERS-clause* を使用する場合、*operand1* は定数にする必要があります。	
operand2	ハンドル名： *operand2* は、親のハンドル名です。	
operand3	ダイアログ **ID**： *operand3* は、ダイアログ作成時に返される一意の識別子です。 これは、フォーマット／長さ I4 で定義する必要があります。	
	ダイアログボックスへのパラメータの引き渡し： ダイアログボックスを開いたときに、パラメータがダイアログボックスに渡されます。	
operand4	*operand4* として、ダイアログボックスに渡されるパラメータを指定します。	
PARAMETERS-clause	パラメータを選択的に渡す： *PARAMETERS-clause* を使用して、パラメータを選択的に渡すことができます。詳細については、以下の「***PARAMETERS** 節*」を参照してください。 **注意:** *operand1* が定数で、ダイアログがカタログされる場合だけ、*PARAMETERS-clause* を使用できます。	
***n*X**	省略されるパラメータの指定： 表記 *n*X を使用して、次の *n* パラメータを省略するように指定できます。例えば、1X は次のパラメータを省略し、3X は次の 3 つのパラメータを省略します。これは、次の *n* パラメータに、ダイアログボックスに渡す値がないことを意味します。省略されるパラメータは、ダイアログボックスの DEFINE DATA PARAMETER ステートメントでキーワード OPTIONAL を使用して定義する必要があります。OPTIONAL は、値を呼び出し側オブジェクトからこのようなパラメータに渡すこともできるということを意味します。	
AD =	属性割り当て： *operand4* が変数の場合は、次のいずれかの方法でマークすることができます。	
	AD=O	変更不可。セッションパラメータ AD=O を参照してください。
	AD=M	変更可。セッションパラメータ AD=M を参照してください。
	AD=A	入力のみ。セッションパラメータ AD=A を参照してください。
	operand4 が定数の場合は、*operand4* を明示的に指定することはできません。定数には常に AD=O が適用されます。	

PARAMETERS 節

```
PARAMETERS {parameter-name = operand4}...
END-PARAMETERS
```

構文要素の説明：

parameter-name	パラメータ名は、ダイアログのパラメータデータエリアセクションに定義されたパラメータの名前です。 **注意:** AD=O でマークされ「参照によって」渡されたパラメータの値がダイアログボックス内で変更されると、ランタイムエラーが発生します。
operand4	*operand4* として、ダイアログボックスに渡されるパラメータを指定します。
END-PARAMETERS	*PARAMETERS-clause* を終了するには、Natural の予約語 END-PARAMETERS を使用する必要があります。

詳細と例

『*プログラミングガイド*』の「*イベントドリブンプログラミング手法*」を参照してください。

91 OPTIONS

```
OPTIONS parameter ...
```

このchapterでは、次のトピックについて説明します。

機能

OPTIONS ステートメントを使用すると、現在の Natural プログラミングオブジェクトに、さまざまなコンパイラオプションをパラメータとして指定できます。同じオプションを、Natural セッション内で COMPOPT システムコマンドを使用して指定できます。

Note: メインフレーム固有でないオプションを使用できます。互換性のために、例えばクロスプラットフォームアプリケーションをプログラミングする場合、このようなオプションはコンパイル時に無視されます。

92 PARSE XML

```
PARSE XML operand1 [INTO [PATH operand2] [NAME operand3] [VALUE operand4]]
 [NORMALIZE] NAMESPACE operand5 PREFIX operand6]
 statement...
END-PARSE                              (structured mode only)
[LOOP]                                 (reporting mode only)
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

PARSE XML ステートメントを使用すると、Natural プログラムから XML ドキュメントを解析できるようになります。『*プログラミングガイド*』の「*インターネットと XML アクセスのステートメント*」も参照してください。

スタティック変数の長さを決定することはできないため、PARSE ステートメントを使用するときはダイナミック変数を使用することをお勧めします。 スタティック変数を使用すると、変数に書き込まれる値が切り捨てられる可能性があります。

Unicode サポートの詳細については、『*Unicode およびコードページのサポート*』ドキュメントの「PARSE XML」を参照してください。

マークアップ

（ASCII ベースシステムで）XML ドキュメントの異なるデータタイプを表すためにパス文字列で使用されるマーキングを次に示します。

マーキング	XML データ	パス文字列内の位置
?	処理命令（<?XML...?> を除く）	末尾
!	コメント	末尾
C	CDATA セクション	末尾
@	属性（メインフレームでは §）	属性名の前
/	終了タグまたはパス内の親名セパレータ	末尾または親名の間
$	解析したデータ - データ文字列	末尾

パス文字列でこの追加のマークアップを使用することにより、出力ドキュメントで XML ドキュメントの各種要素をより容易に識別できます。

グローバル名前空間

グローバル名前空間を指定するには、接頭辞 ":" および空の URI を使用してください。

関連するシステム変数

次の Natural システム変数は、発行された PARSE XML ステートメントごとに自動的に作成されます。

- *PARSE-TYPE
- *PARSE-LEVEL
- *PARSE-ROW
- *PARSE-COL
- *PARSE-NAMESPACE-URI

*PARSE-TYPE、*PARSE-LEVEL、*PARSE-ROW、*PARSE-COL、および *PARSE-NAMESPACE-URI の後の表記 (r) は、PARSE が発行されたステートメントのラベルまたはステートメント番号を示すために使用されます。(r) 指定がない場合、対応するシステム変数は、アクティブな PARSE 処理ループで現在処理されている XML データのシステム変数を表します。

これらのシステム変数の詳細については、『システム変数』ドキュメントを参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S				A	U	B									可	不可
operand2		S				A	U	B									可	可
operand3		S				A	U	B									可	可
operand4		S				A	U	B									可	可
operand5		S	A			A	U	B									可	可
operand6		S	A			A	U	B									可	可

構文要素の説明：

operand1	*operand1* は対象の XML ドキュメントを表します。 XML ドキュメントを解析中に変更することはできません。 解析中に XML ドキュメントを変更（例えば、書き込み）しようとすると、エラーメッセージが表示されます。
operand2	*operand2* は XML ドキュメントのデータの PATH を表します。 PATH には、識別された XML 部分の名前、すべての親の名前、および XML 部分のタイプが含まれます。 **注意:** PATH で指定された情報を使用して、ツリービューを容易に満たすことができます。 「*例 1 - operand2 の使用*」も参照してください。
operand3	*operand3* は XML ドキュメントのデータ要素の NAME を表します。 NAME が値を持たない場合、それに関連するダイナミック変数は *length()=0（空白で満たされたスタティック変数）に設定されます。 「*例 2 - operand3 の使用*」も参照してください。
operand4	*operand4* は XML ドキュメントのデータ要素の内容（VALUE）を表します。 値がない場合、所定のダイナミック変数は *length()=0,（空白で満たされたスタティック変数）に設定されます。 「*例 3 - operand4 の使用*」も参照してください。
operand5 および *operand6* **NORMALIZE NAMESPACE** **PREFIX**	NAMESPACE URI または URI（*operand5*）および名前空間 PREFIX（*operand6*）はランタイム中にコピーされます。 したがって、PARSE XML ループ内の名前空間マッピング配列の修正はパーサに影響しません。 *operand5* および *operand6* はオカレンス数の等しい 1 次元配列です。 名前空間正規化は PARSE ステートメントの機能です。 XML では、要素名に対応する名前空間を定義できます。 `<myns:myentity xmlns:myns="http://myuri" />` NAMESPACE 定義は次の部分で構成されます。 ■ 名前空間 PREFIX（この場合、myns） ■ 名前空間を定義する URI（myuri） 名前空間 PREFIX は要素名の一部です。 つまり、PARSE ステートメントに対して、そして特に *operand2* に対して、生成された PATH 文字列は名前空間 PREFIX に依存します。 Natural プログラム内のパスが特定のタグを示すために使用される場合、XML ドキュメントで正しい NAMESPACE（URI）を異なる PREFIX とともに使用すると失敗します。 名前空間正規化では、すべての名前空間 PREFIX は、NAMESPACE 節で定義されているデフォルト値に設定できます。 最初のエントリは、URI が複数回指定される場合に使用されます。

	XML ドキュメントで複数の PREFIX が使用される場合、最初のものだけが出力に考慮されます。 残りは無視されます。 NAMESPACE 節には、名前空間 URI と接頭辞のペアが含まれます。 次に例を示します。 `uri(1) := 'http://namespaces.softwareag.com/natural/demo'` `pre(1) := 'nat:'` XML ドキュメント内に NAMESPACE が定義されている場合、パーサによってその名前空間（URI）が正規化テーブルに存在するかどうかがチェックされます。正規化テーブルの接頭辞は、XML ドキュメントに定義された名前空間の代わりに、PARSE ステートメントからのすべての出力データに対して使用されます。 以下の項目も参照してください。 ■ *例 4 - operand5 とoperand6 の使用* ■ *例 5 - 名前空間正規化でのoperand5 とoperand6 の使用*
	PREFIX に関する追加情報： また、接頭辞定義には次の点が適用されます。 ■ 名前空間正規化配列の接頭辞定義は、置換される文字列であるため、常にコロン ":" で終了する必要があります。 ■ PREFIX または URI は、名前空間正規化配列には 1 回だけ存在できます。 ■ PREFIX または NAMESPACE URI に末尾の空白が含まれる場合（例えば、スタティック変数の使用時）、外部パーサが呼び出される前に、末尾の空白は削除されます。 ■ 名前空間正規化での PREFIX 定義にコロン ":" のみが含まれる場合、NAMESPACE PREFIX は削除されます。

例

- **例 1 -*operand2* の使用**
- **例 2 - *operand3* の使用**
- **例 3 - *operand4* の使用**
- **例 4 - *operand5* と*operand6* の使用**

例 1 - *operand2* の使用

次のような XML コードがあるとします。

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
         '<employee personnel-id="30016315" >'-
         '<full-name>'-
         '<!--this is just a comment-->'-
         '<first-name>RICHARD</first-name>'-
         '<name>FORDHAM</name>'-
         '</full-name>'-
         '</employee>'
```

これは、次の Natural コードで処理されます。

```
PARSE XML myxml INTO PATH mypath
  PRINT mypath
END-PARSE
```

次の出力を生成します。

```
employee
employee/@personnel-id
employee/full-name
employee/full-name/!
employee/full-name/first-name
employee/full-name/first-name/$
employee/full-name/first-name//
employee/full-name/name
employee/full-name/name/$
employee/full-name/name//
```

```
employee/full-name//
employee//
```

例 2 - *operand3* の使用

次のような XML コードがあるとします。

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
         '<employee personnel-id="30016315" >'-
         '<full-name>'-
         '<!--this is just a comment-->'-
         '<first-name>RICHARD</first-name>'-
         '<name>FORDHAM</name>'-
         '</full-name>'-
         '</employee>'
```

これは、次の Natural コードで処理されます。

```
PARSE XML myxml INTO PATH mypath NAME myname
  DISPLAY (AL=39) mypath myname
END-PARSE
```

Note: 次の出力を生成します。

```
               MYPATH                                MYNAME
-----------------------------------    -------------------------------------


employee                               employee
employee/@personnel-id                 personnel-id
employee/full-name                     full-name
employee/full-name/!
employee/full-name/first-name          first-name
employee/full-name/first-name/$
employee/full-name/first-name//        first-name
employee/full-name/name                name
employee/full-name/name/$
employee/full-name/name//              name
```

```
employee/full-name//                  full-name
employee//                            employee
```

例 3 - *operand4* の使用

次のような XML コードがあるとします。

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
         '<employee personnel-id="30016315" >'-
         '<full-name>'-
         '<!--this is just a comment-->'-
         '<first-name>RICHARD</first-name>'-
         '<name>FORDHAM</name>'-
         '</full-name>'-
         '</employee>'
```

これは、次の Natural コードで処理されます。

```
PARSE XML myxml INTO PATH mypath VALUE myvalue
  DISPLAY (AL=39) mypath myvalue
END-PARSE
```

次の出力を生成します。

```
               MYPATH                                MYVALUE
-----------------------------------    -----------------------------------



employee
employee/@personnel-id                30016315
employee/full-name
employee/full-name/!                  this is just a comment
employee/full-name/first-name
employee/full-name/first-name/$       RICHARD
employee/full-name/first-name//
employee/full-name/name
employee/full-name/name/$             FORDHAM
employee/full-name/name//
```

```
employee/full-name//
employee//
```

例 4 - *operand5* と *operand6* の使用

次のような XML コードがあるとします。

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
         '<nat:employee nat:personnel-id="30016315"'-
         ' xmlns:nat="http://namespaces.softwareag.com/natural/demo">'-
         '<nat:full-Name>'-
         '<nat:first-name>RICHARD</nat:first-name>'-
         '<nat:name>FORDHAM</nat:name>'-
         '</nat:full-Name>'-
         '</nat:employee>'
```

これは、次の Natural コードで処理されます。

```
PARSE XML myxml INTO PATH mypath
  PRINT mypath
END-PARSE
```

次の出力を生成します。

```
nat:employee
nat:employee/@nat:personnel-id
nat:employee/@xmlns:nat
nat:employee/nat:full-Name
nat:employee/nat:full-Name/nat:first-name
nat:employee/nat:full-Name/nat:first-name/$
nat:employee/nat:full-Name/nat:first-name//
nat:employee/nat:full-Name/nat:name
nat:employee/nat:full-Name/nat:name/$
nat:employee/nat:full-Name/nat:name//
```

```
nat:employee/nat:full-Name//
nat:employee//
```

例 5 - 名前空間正規化での*operand5* と*operand6* の使用

NORMALIZE NAMESPACE を使用すると、例 4 のような NAMESPACE PREFIX の異なる同じ XML ドキュメントでまったく同じ出力が生成されます。

XML コード：

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
         '<natural:employee natural:personnel-id="30016315"'-
         ' xmlns:natural="http://namespaces.softwareag.com/natural/demo">'-
         '<natural:full-Name>'-
         '<natural:first-name>RICHARD</natural:first-name>'-
         '<natural:name>FORDHAM</natural:name>'-
         '</natural:full-Name>'-
         '</natural:employee>'
```

Natural コード：

```
uri(1) := 'http://namespaces.softwareag.com/natural/demo'
pre(1) := 'nat:'
*
PARSE XML myxml INTO PATH mypath NORMALIZE NAMESPACE uri(*) PREFIX pre(*)
  PRINT mypath
END-PARSE
```

上記プログラムの出力：

```
nat:employee
nat:employee/@nat:personnel-id
nat:employee/@xmlns:nat
nat:employee/nat:full-Name
nat:employee/nat:full-Name/nat:first-name
nat:employee/nat:full-Name/nat:first-name/$
nat:employee/nat:full-Name/nat:first-name//
nat:employee/nat:full-Name/nat:name
nat:employee/nat:full-Name/nat:name/$
nat:employee/nat:full-Name/nat:name//
nat:employee/nat:full-Name//
nat:employee//
```

93 PASSW

```
PASSW=operand1
```

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **HISTOGRAM** | **GET** | **GET SAME** | **GET TRANSACTION** | **LIMIT** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

PASSW ステートメントは、パスワード保護された Adabas または VSAM ファイルへのアクセスに必要なデフォルトのパスワードを指定するために使用します。

Note: このパスワードは、データベースアクセスステートメント FIND、GET、HISTOGRAM、READ、STORE の PASSWORD 節を使用して上書きできます。

Natural Security の考慮事項

ライブラリのセキュリティプロファイルで、デフォルトの Adabas パスワード（『*Natural Security*』ドキュメントを参照）を指定できます。このパスワードは、個々パスワードも PASSW ステートメントも指定されていないすべてのデータベースアクセスステートメントに適用されます。 これは、パスワードが指定されたセキュリティプロファイルを持つライブラリ内に適用されます。また、パスワードが指定されていないセキュリティプロファイルを持つ別のライブラリに後でログオンするときにも引き続き有効です

構文説明

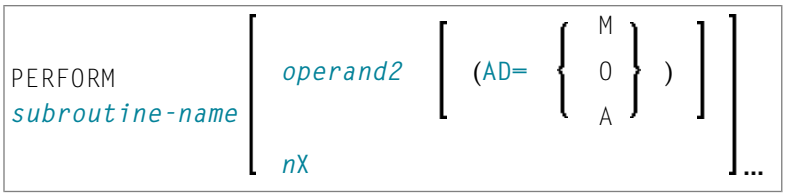
オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S				A											可	不可

構文要素の説明：

operand1	パスワード： パスワード（*operand1*）は、英数字定数または英数字変数の内容として指定できます。最大 8 文字から成り、特殊文字または埋め込み空白は使用できません。 パスワードを定数として指定する場合は、一重引用符で囲む必要があります。 PASSW ステートメントで指定したパスワードは、個々のパスワードが指定されていないすべてのデータベースアクセスステートメント（FIND、GET、HISTOGRAM、READ、STORE）に適用されます。 一度指定したパスワードは、次の PASSW ステートメントの実行で別のパスワードが指定されるまで、または Natural セッションが終了するまで有効です。 特定のデータベースアクセスステートメントで指定したパスワードは、そのステートメントにのみ適用され、他の後続のステートメントには適用されません。

94 PERFORM



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CALL** | **CALL FILE** | **CALL LOOP** | **CALLNAT** | **DEFINE SUBROUTINE** | **ESCAPE** | **FETCH**

関連機能グループ：「*プログラムおよびルーチンの呼び出し*」

機能

PERFORM ステートメントは、Natural サブルーチンを呼び出すために使用します。

PERFORM ステートメントのネスト構造

呼び出し先サブルーチン内で他のサブルーチンを呼び出すための PERFORM ステートメントを指定することもできます（ネストのレベル数は、要求されるメモリのサイズによって制限されます）。

サブルーチンは自分自身を呼び出すことができます（再帰サブルーチン）。 再帰的に呼び出される外部サブルーチン内にデータベース処理が含まれている場合、Naturalはデータベース処理が論理的に分割されるようにします。

ダイナミック変数を使用したパラメータ引き渡し

CALLNAT ステートメントを参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												
operand2	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	O

ステートメント参照	ダイナミック定義
可	可

構文要素の説明：

subroutine-name	呼び出すサブルーチン： サブルーチン名（最大 32 文字）については、ユーザー定義変数と同じ命名規則が適用されます。 サブルーチン名は、該当のサブルーチンが定義されているモジュールの名前に依存しません（同じ名前にすることは可能ですが、同じ名前にする必要はありません）。 呼び出すサブルーチンは、DEFINE SUBROUTINE ステートメントで定義しておく必要があります。 インラインサブルーチンまたは外部サブルーチンとして指定できます（DEFINE SUBROUTINE ステートメントを参照）。 1 つのオブジェクト内で 50 個までの外部サブルーチンを参照できます。 サブルーチンで使用可能なデータ ■ インラインサブルーチン 呼び出し側オブジェクトからインラインサブルーチンに明示パラメータを渡すことはできません。 インラインサブルーチンは、同一オブジェクトモジュール内に定義されたローカルデータエリアや現在設定されているグローバルデータエリアにアクセスできます。 ■ 外部サブルーチン 外部サブルーチンは、現在設定されているグローバルデータエリアにアクセスできます。 さらに、PERFORM ステートメントで呼び出し側オブジェクトから外部サブルーチンにパラメータを渡すことができるため（*operand2*を参照）、グローバルデータエリアのサイズを削減できます。
operand2	外部サブルーチンへのパラメータの引き渡し： PERFORM ステートメントで外部サブルーチンを呼び出すとき、PERFORM ステートメントで呼び出し側オブジェクトから外部サブルーチンに 1 つ以上のパラメータ（*operand2*）を渡すことができます。 インラインサブルーチンでは、*operand2*は指定できません。 パラメータを渡す場合、DEFINE DATA ステートメントでパラメータリストの構造を定義する必要があります。

	デフォルトでは、パラメータは「参照によって」渡されます。つまり、データはアドレスパラメータを介して転送され、パラメータ値自体は移動されません。ただし、パラメータを「値によって」渡す、つまり実際のパラメータ値を渡すこともできます。そのためには、これらのフィールドをサブルーチンのDEFINE DATA PARAMETERステートメントにBY VALUEまたはBY VALUE RESULTオプション付きで定義します。 ■ パラメータが「参照によって」渡される場合、次のことが適用されます。呼び出し側オブジェクトのパラメータの順序、フォーマット、および長さは、呼び出し先サブルーチンのDEFINE DATA PARAMETER構造の順序、フォーマット、および長さと正確に一致している必要があります。変数の名前が呼び出し側オブジェクトとサブルーチンで異なる場合があります。 ■ パラメータが「値によって」渡される場合、次のことが適用されます。呼び出し側オブジェクトのパラメータの順序は、呼び出し先サブルーチンのDEFINE DATA PARAMETER構造の順序と正確に一致している必要があります。呼び出し側オブジェクトの変数のフォーマットと長さは、サブルーチンの変数と異なっていてもかまいません。ただし、データ転送の互換性が必要です。変数の名前が呼び出し側オブジェクトとサブルーチンで異なる場合があります。サブルーチンで変更されたパラメータ値を呼び出し側オブジェクトに戻す場合、これらのフィールドをBY VALUE RESULT付きで定義する必要があります。BY VALUE（RESULTなし）では、AD指定（下記も参照）に関係なく、変更したパラメータ値を呼び出し側オブジェクトに戻すことはできません。 **注意:** BY VALUEでは、パラメータ値の内部コピーが作成されます。サブルーチンはこのコピーにアクセスし、それを変更できますが、その変更は呼び出し側オブジェクトの元のパラメータ値には影響しません。BY VALUE RESULTでは、同様に内部コピーが作成されます。ただし、サブルーチンの終了後、元のパラメータ値はコピーの（変更された）値で上書きされます。 いずれのパラメータ引き渡し方法にも、次が適用されます。 ■ 呼び出し先サブルーチンのパラメータデータエリアでは、グループの再定義はREDEFINEブロック内でのみ許可されます。 ■ 配列を渡す場合、サブルーチンのパラメータデータエリア内の次元数とオカレンス数はPERFORMパラメータリストと同じにする必要があります。 **注意:** インデックス付きグループの一部として定義された配列の複数オカレンスをPERFORMステートメントで渡す場合、サブルーチンのパラメータデータエリア内の対応するフィールドを再定義しないでください。再定義すると、不正なアドレスが渡されます。
AD=	属性の定義： *operand2*が変数の場合は、次のいずれかの方法でマークすることができます。
	AD=O — 変更不可。セッションパラメータAD=Oを参照してください。 **注意:** 内部的に、AD=OはBY VALUEと同じ方法で処理されます（*operand2*の注を参照）。

	AD=M	変更可。セッションパラメータ AD=M を参照してください。 これはデフォルト設定です。
	AD=A	入力のみ。セッションパラメータ AD=A を参照してください。
	operand2 が定数の場合は、AD を明示的に指定することはできません。 定数には常に AD=O が適用されます。	
***n*X**	省略されるパラメータの指定： 表記 *n*X を使用して、次の *n* パラメータを省略するように指定できます。例えば、1X は次のパラメータを省略し、3X は次の 3 つのパラメータを省略します。これは、次の *n* パラメータに、外部サブルーチンに渡す値がないことを意味します。 省略されるパラメータは、サブルーチンの DEFINE DATA PARAMETER ステートメントでキーワード OPTIONAL を使用して定義する必要があります。 OPTIONAL は、値を呼び出し側オブジェクトからこのようなパラメータに渡すこともできるということを意味します。	

例

- 例 1 - インラインサブルーチンとしての PERFORM
- 例 2 - 外部サブルーチンとしての PERFORM

例 1 - インラインサブルーチンとしての PERFORM

```
** Example 'PEREX1': PERFORM   (as inline subroutine)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE  (A20/2)
  2 PHONE
*
1 #ARRAY    (A75/1:4)
1 REDEFINE #ARRAY
  2 #ALINE  (A25/1:4,1:3)
1 #X        (N2) INIT <1>
1 #Y        (N2) INIT <1>
END-DEFINE
*
LIMIT 5
FIND EMPLOY-VIEW WITH CITY = 'BALTIMORE'
  MOVE NAME             TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
```

```
  MOVE PHONE           TO #ALINE (#X+3,#Y)
  IF  #Y = 3
    RESET INITIAL #Y
    /*
    PERFORM PRINT
    /*
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
    /*
    PERFORM PRINT
    /*
  END-ENDDATA
END-FIND
*
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
END-SUBROUTINE
*
END
```

プログラム **PEREX1** の出力：

```

JENSON                    LAWLER                    FORREST
2120 HASSELL              4588 CANDLEBERRY AVE      37 TENNYSON DRIVE
 #206                     BALTIMORE                 BALTIMORE
998-5038                  629-0403                  881-3609

ALEXANDER                 NEEDHAM
409 SENECA DRIVE          12609 BUILDERS LANE
BALTIMORE                 BALTIMORE
345-3690                  641-9789
```

例 2 - 外部サブルーチンとしての PERFORM

PERFORM ステートメントを含むプログラム：

```
** Example 'PEREX2': PERFORM (as external subroutine)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE  (A20/2)
  2 PHONE
*
1 #ALINE   (A25/1:4,1:3)
1 #X       (N2)            INIT <1>
```

```
1 #Y       (N2)            INIT <1>
END-DEFINE
*
LIMIT 5
*
FIND EMPLOY-VIEW WITH CITY = 'BALTIMORE'
  MOVE NAME            TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE           TO #ALINE (#X+3,#Y)
  IF  #Y = 3
    RESET INITIAL #Y
    /*
    PERFORM PEREX2E #ALINE(*,*)
    /*
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
    /*
    PERFORM PEREX2E #ALINE(*,*)
    /*
  END-ENDDATA
END-FIND
*
END
```

プログラム **PEREX2** から呼び出されるパラメータを含む外部サブルーチン **PEREX3**：

```
** Example 'PEREX3': SUBROUTINE (external subroutine with parameters)
************************************************************************
DEFINE DATA
PARAMETER
1 #ALINE  (A25/1:4,1:3)
END-DEFINE
*
DEFINE SUBROUTINE PEREX2E
  WRITE NOTITLE (AD=OI) #ALINE(*,*)
  RESET #ALINE(*,*)
  SKIP 1
END-SUBROUTINE
*
END
```

プログラム **PEREX2** の出力：

```
JENSON                    LAWLER                    FORREST
2120 HASSELL              4588 CANDLEBERRY AVE      37 TENNYSON DRIVE
 #206                     BALTIMORE                 BALTIMORE
998-5038                  629-0403                  881-3609

ALEXANDER                 NEEDHAM
409 SENECA DRIVE          12609 BUILDERS LANE
BALTIMORE                 BALTIMORE
345-3690                  641-9789
```

95 PERFORM BREAK PROCESSING

```
PERFORM BREAK [PROCESSING] [(r)]
AT BREAK statement ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **READ** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

PERFORM BREAK PROCESSING ステートメントは、自動ブレイク処理が設定されない FOR、REPEAT、CALL LOOP、および CALL FILE ステートメントで生成されるループ内にブレイク処理を設定するために使用します。または、ユーザー開始のブレイク処理が必要な場合に使用します。 レコードが読み取られた直後に実行される自動ブレイク処理とは異なり、PERFORM BREAK PROCESSING ステートメントは、プログラムの通常の流れの中で出現したときに初めて実行されます。

このステートメントでは、（コントロールフィールドの値に基づいて）ブレイク処理の条件をチェックし、Naturalシステム関数を評価します。このチェックおよびシステム関数の評価は、実行対象のステートメントが出現するたびに行われます。 このステートメントは、IF ステートメントで指定した条件に応じて実行することもできます。

構文説明

(*r*)	ステートメント参照表記： デフォルトでは、プログラム、サブプログラム、またはサブルーチンの実行終了時に最後の PERFORM BREAK 条件が真となります。 表記 (*r*) を使用して、PERFORM BREAK の最後の処理を特定のループに関連付けることができます。 この場合、PERFORM BREAK はそのループの終了処理の中（最後の自動 BREAK 処理後かつ AT END OF DATA ステートメントの実行前）で実行されます。
AT BREAK *statement...*	AT BREAK ステートメントの構文を参照してください。

例

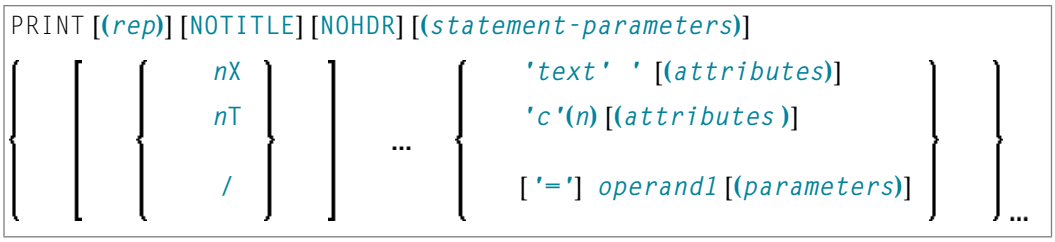
```
** Example 'PBPEX1S': PERFORM BREAK PROCESSING  (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 #INDEX (N2)
1 #LINE  (N2) INIT <1>
END-DEFINE
*
FOR #INDEX 1 TO 18
  PERFORM BREAK PROCESSING
  /*
  AT BREAK OF #INDEX /1/
    WRITE NOTITLE / 'PLEASE COMPLETE LINES 1-9 ABOVE' /
    RESET INITIAL #LINE
  END-BREAK
  /*
  WRITE NOTITLE '_' (64) '=' #LINE
  ADD 1 TO #LINE
END-FOR
*
END
```

プログラム **PBPEX1S** の出力：

```
                                                                 #LINE:   1
                                                                 #LINE:   2
                                                                 #LINE:   3
                                                                 #LINE:   4
                                                                 #LINE:   5
                                                                 #LINE:   6
                                                                 #LINE:   7
                                                                 #LINE:   8
                                                                 #LINE:   9

PLEASE COMPLETE LINES 1-9 ABOVE

                                                                 #LINE:   1
                                                                 #LINE:   2
                                                                 #LINE:   3
                                                                 #LINE:   4
                                                                 #LINE:   5
                                                                 #LINE:   6
                                                                 #LINE:   7
                                                                 #LINE:   8
                                                                 #LINE:   9

PLEASE COMPLETE LINES 1-9 ABOVE
```

レポーティングモードの例はライブラリ SYSEXRM のプログラム **PBPEX1R** を参照してください。

96 PRINT



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「構文記号」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

PRINT ステートメントは、出力をフリーフォーマットで生成するために使用します。

PRINT ステートメントは、次の点で WRITE ステートメントと異なります。

- 各オペランドの出力は、オペランドの長さではなく値の内容に合わせて書き込まれます。 数値の先頭のゼロおよび英数字の末尾の空白は省略されます。 セッションパラメータ AD では、数値を左揃えで出力するか右揃えで出力するかを定義します。 AD=L の場合、数値の末尾の空白は省略されます。 AD=R の場合、数値の先頭の空白は出力されます。
- 出力結果が現在の行サイズ（LS パラメータ）を超過した場合、出力が次の行に継続します。 このとき、英数字定数、または編集マスク指定のない英数字変数の内容は、現在の行の右端の空白または英字でも数字でもない文字で分割されます。 分割された値の初めの部分は現在の行に出力され、後ろの部分は次の行に出力されます。 他のすべてのオペランドについては、値全体が次の行に出力されます。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1		S	A	G	N	A	U	N	P	I	F	B	D	T	L		G	O	可	不可

構文要素の説明：

(*rep*)	レポート指定： 表記 (*rep*) を使用して、PRINT ステートメントを適用するレポートの ID を指定できます。 範囲 0～31 の値、または DEFINE PRINTER ステートメントを使用して割り当てた論理名を指定できます。 (*rep*) 指定がない場合、PRINT ステートメントは最初のレポート（レポート 0）に適用されます。 このプリンタファイルが PC として Natural に定義されている場合、レポートは PC にダウンロードされます。*例 2* を参照してください。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。
NOTITLE	デフォルトページタイトルの省略： Natural は、PRINT ステートメントによって出力される各ページに 1 行のタイトル行を生成します。 このタイトルには、ページ番号と日時が含まれます。時刻は、セッション開始時（TP モード）またはジョブ開始時（バッチモード）に設定されます。 このデフォルトのタイトル行は WRITE TITLE ステートメントを使用して上書きできます。または、PRINT ステートメントに NOTITLE 節を指定することで省略することもできます。 例： ■ デフォルトタイトルが生成されます。 `PRINT NAME`

	■ ユーザータイトルが生成されます。 `PRINT NAME WRITE TITLE 'user-title'` ■ タイトルは生成されません。 `PRINT NOTITLE NAME` NOTITLEオプションを使用すると、同じオブジェクト内でデータを同じレポートに書き込むすべてのDISPLAY、PRINT、およびWRITEステートメントに適用されます。
NOHDR	列ヘッダーの省略： PRINTステートメント自体では、列ヘッダーは生成されません。 ただし、PRINTステートメントをDISPLAYステートメントと組み合わせて使用する場合、PRINTステートメントのNOHDRオプションを使用して、DISPLAYステートメントによって生成される列ヘッダーを省略できます。 NOHDRオプションは、PRINTステートメントの実行によって新しいページが出力される場合にのみ有効です。 NOHDRオプションを使用しないと、この新しいページにDISPLAYステートメントの列ヘッダー（存在する場合）が出力されます。NOHDRを使用すると、出力されません。
statement-parameters	ステートメントレベルでのパラメータ定義： 1つまたは複数のパラメータをカッコで囲んで、ステートメントレベルで、つまりPRINTステートメントまたは出力要素の直後に指定できます。 この方法で指定した各パラメータは、以前にGLOBALSコマンド、SET GLOBALSステートメント（レポーティングモードでのみ）、またはFORMATステートメントで指定したパラメータを上書きします。 複数のパラメータを指定する場合は、各パラメータ間を1つ以上の空白で区切る必要があります。1つのパラメータエントリを2行に分けて指定することはできません。 ここで適用されるパラメータ設定は、変数フィールドにのみ関連し、テキスト定数には影響しません。 テキスト定数にフィールド属性を設定する場合は、この要素に属性を明示的に設定する必要があります（「*要素（フィールド）レベルでのパラメータ定義*」を参照）。 以下の項目も参照してください。 ■ *パラメータのリスト* ■ *ステートメントおよび要素（フィールド）レベルでのパラメータ使用例*
n*X, *n*T, */	下記の「*フィールドの位置指定、テキスト／属性の割り当て*」を参照してください。

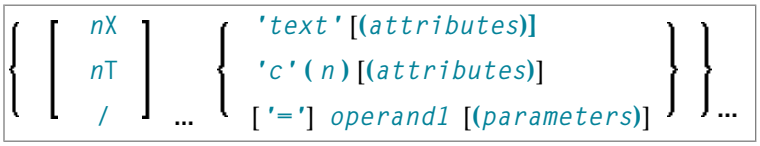
パラメータのリスト

PRINT ステートメントで指定可能なパラメータ		指定（S = ステートメントレベル、E = 要素レベル）
AD	属性定義	SE
AL	出力の英数字長	SE
CD	カラー定義	SE
CV	制御変数	SE
DF	日付フォーマット	SE
DL	出力の表示長	SE
DY	ダイナミック属性	SE
EM	編集マスク	SE
EMU	Unicode 編集マスク	E
FL	浮動小数点仮数長	SE
MC	マルチプルバリューフィールド数	S
MP	レポートの最大ページ数	S
NL	出力の数値長	SE
PC	ピリオディックグループ数	S
PM	出力モード	SE
SG	符号の位置	SE
ZP	ゼロ出力	SE

各セッションパラメータの詳細については、『パラメータリファレンス』を参照してください。

ステートメントおよび要素（フィールド）レベルでのパラメータ使用例

フィールドの位置指定、テキスト／属性の割り当て



フィールドの位置指定表記

*n***X**	列の間隔： この表記により、列の間に *n* 個のスペースが挿入されます。。 `PRINT NAME 5X SALARY`
*n***T**	タブ設定： *n*T 表記により、位置 *n* を出力するように位置指定（タブ設定）されます。 後方への位置指定は行送りになります。 次の例では、NAME は位置 25 から、SALARY は位置 50 から出力されます。 `PRINT 25T NAME 50T SALARY`
/	行送り - スラッシュ表記： フィールドまたはテキスト要素の間に配置した場合は、"/" により次の出力行の先頭に位置指定されます。 `PRINT NAME / SALARY`

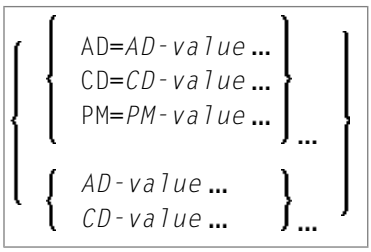
テキスト／属性割り当て

'*text*'	テキスト割り当て： 一重引用符で囲まれた文字列が表示されます。 `PRINT 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT`

'*c*' (*n*)	文字の繰り返し： フィールド値の直前に、一重引用符で囲まれた文字が *n* 回表示されます。 `PRINT '*' (5) '=' NAME`
'='	フィールドヘッダーの後に位置するフィールド内容： フィールドの前に配置した場合は、等号 '=' により、DEFINE DATA ステートメントまたは DDM の定義に従って、フィールドヘッダーはフィールド内容の前に表示されます。 `PRINT '=' NAME`
operand1	出力するフィールド： *operand1* として、出力するフィールドを指定します。
parameters	要素（フィールド）レベルでのパラメータ定義： 1つまたは複数のパラメータ（上記の表を参照）をカッコで囲んで、*operand1* の直後に指定できます。 この方法で指定した各パラメータは、以前にステートメントレベル、GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードでのみ）、または FORMAT ステートメントで指定したパラメータを上書きします。 複数のパラメータを指定する場合は、各エントリ間に 1 つ以上の空白を配置する必要があります。 エントリを 2 行のステートメント行に分割することはできません。 以下の項目も参照してください。 ■ *ステートメントのパラメータ* ■ *ステートメントおよび要素（フィールド）レベルでのパラメータ使用例*

出力属性

attributes は、テキスト表示に使用される出力属性を示します。 可能な属性：



指定可能なセッションパラメータ値については、『*パラメータリファレンス*』ドキュメントの該当するセクションを参照してください。

- 「*AD - 属性定義*」の「*フィールド表現*」
- *CD - カラー定義*
- *PM - 出力モード*

Note: コンパイラは、実際には1つの出力フィールドに複数の属性値を受け入れます。例えば、「AD=BDI」と指定できます。ただし、この場合は最後の値のみが適用されます。示した例では、値 "I" のみが有効になり、出力フィールドは強調表示されます。

例

例 1 - PRINT ステートメント

```
** Example 'PRTEX1': PRINT
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
  2 ADDRESS-LINE (2)
END-DEFINE
*
LIMIT 1
READ EMPLOY-VIEW BY CITY
  /*
  WRITE NOTITLE 'EXAMPLE 1:'
              // 'RESULT OF WRITE STATEMENT:'
  WRITE        /  NAME  ',' FIRST-NAME ':' JOB-TITLE '*' (30)
  WRITE        / 'RESULT OF PRINT STATEMENT:'
  PRINT        /  NAME  ',' FIRST-NAME ':' JOB-TITLE '*' (30)
  /*
  WRITE       // 'EXAMPLE 2:'
              // 'RESULT OF WRITE STATEMENT:'
  WRITE        /  NAME 60X ADDRESS-LINE (1:2)
  WRITE        / 'RESULT OF PRINT STATEMENT:'
  PRINT        /  NAME 60X ADDRESS-LINE (1:2)
  /*
```

```
END-READ
END
```

プログラム **PRTXEX1** の出力：

```
EXAMPLE 1:

RESULT OF WRITE STATEMENT:

SENKO                , WILLIE               : PROGRAMMER
*****************************

RESULT OF PRINT STATEMENT:

SENKO , WILLIE : PROGRAMMER *****************************


EXAMPLE 2:

RESULT OF WRITE STATEMENT:

SENKO
2200 COLUMBIA PIKE    #914

RESULT OF PRINT STATEMENT:

SENKO                                                              2200 COLUMBIA
PIKE  #914
```

例 2 - レポートが PC にダウンロードされる PRINT ステートメント

```
** Example 'PCPIEX1': PRINT to PC
**
** NOTE: Example requires that Natural Connection is installed.
************************************************************************
DEFINE DATA LOCAL
01 PERS VIEW OF EMPLOYEES
  02 PERSONNEL-ID
  02 NAME
  02 CITY
END-DEFINE
*
FIND PERS WITH CITY = 'NEW YORK'                 /* Data selection
  PRINT (7) 5T CITY 20T NAME 40T PERSONNEL-ID     /* (7) designates
                                                  /* the output file
                                                  /* (here the PC).
END-FIND
END
```

97 PROCESS

```
PROCESS view-name USING operand1=operand2 GIVING operand3...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

PROCESS ステートメントは、Entire System Server と組み合わせて使用します。 Entire System Server は、ファイルの読み書き、VTOC およびカタログ管理、JES キューなどの各種オペレーティングシステム機能の使用を可能にします。

PROCESS ステートメントと個々の節の詳細については、『*Entire System Server User's Guide*』の「*Getting Started*」セクションを参照してください。

制限事項

このステートメントは Entire System Server 以外では使用できません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S				A		N	P			B						可	不可
operand2	C	S				A	U	N	P			B						可	不可
operand3		S				A		N	P			B						可	不可

構文要素の説明：

view-name	Entire System Server で使用されるビューの名前です。
USING	USING 節は、Entire System Server プロセッサにパラメータを渡すために使用します。 そのためには、Entire System Server に定義されたビューで値（*operand2*）をフィールド（*operand1*）に割り当てます。 ビューについては、Entire System Server ドキュメントを参照してください。

	注意: "*operand1*=*operand2*" の複数指定は、INPUT 区切り文字（セッションパラメータ ID で指定）またはコンマで区切る必要があります。 セッションパラメータ DC でコンマを小数点文字として定義している場合は、この目的でコンマを使用することはできません。
GIVING	GIVING 節は、値が Entire System Server プロセッサによって返されるフィールド（*operand3*）を指定するために使用します。 各フィールドは、Entire System Server で使用されるビューに定義する必要があります。

98 PROCESS COMMAND

ストラクチャードモード構文

```
PROCESS COMMAND ACTION
      ┌          CLOSE                                                    ┐
      │          CHECK                                                    │
      │ ┌              ┐                        PROCESSOR-NAME=operand1   │
      │ │    EXEC      │          USING         COMMAND-LINE              │
      │ │              │                        (index[:index])=operand2  │
      │ │    TEXT      │                                                  │
      │ └              ┘                                                  │
      │          HELP                                                     │
      └          GET USING   PROCESSOR-NAME=operand1                     ┘
                             GETSET-FIELD-NAME=operand3
                 SET USING   PROCESSOR-NAME=operand1
                             GETSET-FIELD-NAME=operand3

                             GETSET-FIELD-VALUE=operand4
```

レポーティングモード構文

```
PROCESS COMMAND ACTION
      ┌          CLOSE [GIVING NATURAL-ERROR]                                          ┐
      │          CHECK                                                                 │
      │ ┌              ┐                        PROCESSOR-NAME=operand1                │
      │ │    EXEC      │          USING         COMMAND-LINE (index[:                  │
      │ │              │                        index])=operand2                       │
      │ │    TEXT      │                        GIVING RESULT-FIELD (index[:           │
      │ └              ┘                        index ])                               │
      │          HELP                          RETURN-CODE [NATURAL-ERROR]            │
      └          GET USING   PROCESSOR-NAME=operand1                                  ┘
                             GETSET-FIELD-NAME=operand3
                             GETSET-FIELD-VALUE=operand4 [NATURAL-ERROR]
                 SET USING   PROCESSOR-NAME=operand1
                             GETSET-FIELD-NAME=operand3
                             GETSET-FIELD-VALUE=operand4 [GIVING
                             NATURAL-ERROR]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

Natural ユーティリティ SYSNCP で作成したコマンドプロセッサは、PROCESS COMMAND ステートメントを使用して Natural プログラムから呼び出すことができます。

Natural コマンドプロセッサの作成方法の詳細については、『SYSNCP ユーティリティ』ドキュメントを参照してください。

Note: PROCESS COMMAND ステートメント内の単語 「COMMAND」 は実際のビュー名です。 使用されるビューの名前は必ずしも 「COMMAND」 である必要はありませんが、同じ名前の **DDM** が存在するため、「COMMAND」 を使用することをお勧めします。 DEFINE DATA ステートメント（例えば、COMMAND VIEW OF COMMAND）内でこの DDM を参照する必要があります。

セキュリティに関する考慮事項

Natural Security では、コマンドプロセッサで定義された特定のキーワードや機能の使用を制限できます。 特定のユーザーまたはユーザーグループに対して、キーワードや機能の使用を許可／禁止できます。 詳細については、『*Natural Security*』ドキュメントを参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S				A											不可	不可
operand2	C	S	A	G		A		N									不可	不可
operand3	C	S				A		N									不可	不可
operand4	C	S				A		N	P	I							不可	不可

構文要素の説明：

CLOSE	CLOSE では、コマンドプロセッサの使用を終了し、コマンドプロセッサバッファを解放します。 セッション中にコマンドプロセッサを使用し、CLOSE で解放しないと、ユーザーのスレッド中に NCPWORK という名前のバッファが存在します。 コマンドプロセッサのランタイム部分でこのバッファが必要になります。このバッファは PROCESS COMMAND ACTION CLOSE ステートメントで解放できます。 このステートメントに続けて PROCESS COMMAND ステートメントを指定すると、コマンドプロセッサバッファが再びオープンされます。

	「*例 1 - PROCESS COMMAND ACTION CLOSE*」も参照してください。
CHECK	CHECK は、コマンドが PROCESS COMMAND EXEC ステートメントで実行可能かどうかを判断するための手段として使用します。 指定プロセッサ名に対して、ランタイムチェックが 2 段階で行われます。 ■ 現在のライブラリまたは STEPLIB の 1 つにプロセッサが存在するかどうかをチェックします。 ■ コマンド行 COMMAND-LINE (1) の内容が受け入れ可能かどうかを判断するために分析します。 または、ランタイムアクション定義 "R"、"M"、および "1〜9" は RESULT-FIELD (1:9) に書き込まれます。 NATURAL-ERROR フィールドをビューまたは GIVING 節で指定した場合、エラーコードが返されます。 このフィールドが利用できず、コマンド分析が失敗すると、Natural システムエラーが発生します。 **注意:** CHECK オプションの機能は EXEC オプションの一部としても実行されるため、各 EXEC の前に CHECK を使用する必要はありません。
EXEC	EXEC の動作は CHECK とまったく同じですが、それに加え、ランタイムアクションエディタで指定されたランタイムアクションが実行されます。 COMMAND-LINE (1) のみが必要です。 RESULT-FIELD のオカレンスは 9 個まで使用できます（ただし、最適なパフォーマンスを得るためには、実際に必要なオカレンスのみを使用してください）。 **注意:** EXEC は、現在アクティブなプログラムから抜けるために使用できる唯一のオプションです。 ランタイムアクション定義に FETCH または STOP ステートメントが含まれる場合が該当します。 「*例 2 - PROCESS COMMAND ACTION EXEC*」も参照してください。
HELP	HELP は、有効なすべてのキーワード、同義語、およびオンラインヘルプウィンドウの作成などの機能のリストを返します。 このリストは RESULT-FIELD のフィールド（複数可）に含まれます。返されるヘルプのタイプはコマンド行の内容に依存します。 ■ COMMAND-LINE (1) には検索条件を含める必要があります。 ■ COMMAND-LINE (2)（指定する場合）には開始値または検索値を含める必要があります。 ■ COMMAND-LINE (3)（指定する場合）には開始値を含める必要があります。 詳細については、以下のセクションを参照してください。 ■ *キーワードの HELP* ■ *同義語の HELP* ■ *グローバル機能の HELP* ■ *ローカル機能の HELP* ■ *IKN の HELP* ■ *IFN の HELP*

	注意: 最適なパフォーマンスを得るためには、フィールド RESULT-FIELD のオカレンス数は、画面に表示する行数を超過しないようにする必要があります。 最低 1 オカレンスを使用する必要があります。
TEXT	TEXT オプションは、プロセッサの全般情報およびキーワードや機能に関連するテキストを提供するために使用します。 このテキストは、コマンドプロセッサ定義時に SYSNCP ユーティリティのキーワードエディタやアクションエディタに入力されたものと同じです。 詳細については、以下のセクションを参照してください。 ■ *全般的な情報の **TEXT*** ■ *キーワード情報の **TEXT*** ■ *機能情報の **TEXT*** **注意:** キーワードや機能に関するテキストにアクセスするには、SYSNCP ユーティリティのプロセッサヘッダーメンテナンス 3 画面の Catalog user texts フィールドに "Y" が指定されている必要があります。

キーワードの HELP

このオプションは、アルファベット順にソートされたキーワードまたは同義語を IKN 付きで返します。

コマンド行	内容	
1	インジケータ "K" で始める必要があります。	
	次のキーワードタイプが返されます。	
	*	すべてのタイプのキーワード
	1	タイプ "1" のキーワード
	2	タイプ "2" のキーワード
	3	タイプ "3" のキーワード
	P	タイプ "P"（パラメータ）のキーワード
	オプション：	
	I	キーワードと IKN を返します。
	T	キーワードの一部を大文字で示します（可能な省略形を示すため）。
	S	同義語とキーワードを返します。
	X	指定したキーワードの同義語のみを返します。
	A	内部キーワードも返されます。
	+	検索には開始値は含まれません。

コマンド行	内容
2	キーワード検索の開始値（オプション）。 デフォルトでは、検索は開始値から始まります。 ただし、オプション "+" を指定すると、開始値自体は検索に含まれず、次に高い値から検索されます。

フィールド `RESULT-FIELD (1:n)` は、指定したリストを返します。

例：

```
Command Line 1:    K*X     Returns all synonyms of all keyword types.
```

```
Command Line 1:    K123S   Returns all keywords of type 1, 2 and 3 including synonyms.
```

同義語の HELP

指定された IKN に対して、このオプションは元のキーワードとすべての同義語を返します。

コマンド行	内容	
1	インジケータ "S" で始める必要があります。	
	オプション：	
	T	キーワードの一部を大文字で示します（可能な省略形を示すため）。
2	フォーマット N4 のキーワードの内部キーワード番号（IKN）。	

フィールド `RESULT-FIELD (1)` は、元のキーワードを返します。 フィールド `RESULT-FIELD (2:n)` は、このキーワードに関連する同義語を返します。

例：

入力：	出力：
`Command Line 1:    S` `Command Line 2:    1003`	`Result-Field 1:    Edit` `Result-Field 2:    Maintain` `Result-Field 3:    Modify`

グローバル機能の HELP

このオプションは、すべてのグローバル機能のリストを返します。

コマンド行	内容	
1	インジケータ "G" で始める必要があります。	
	オプション：	
	I	内部機能番号（IFN：Internal Function Number）も返されます。
	T	キーワードの一部を大文字で示します（可能な省略形を示すため）。
	S	RESULT-FIELD に返されたキーワードは列に整列されます。
	A	内部キーワードも返されます。
	1	タイプ 1 の特定のキーワードを含む機能のみが返されます。
	2	タイプ 2 の特定のキーワードを含む機能のみが返されます。
	3	タイプ 3 の特定のキーワードを含む機能のみが返されます。
	+	検索には開始値は含まれません。
2	グローバル機能検索の開始値。 キーワードは "123" の順序で指定する必要があります。 デフォルトでは、検索は開始値から始まります。 ただし、オプション "+" を指定すると、開始値自体は検索に含まれず、次に高い値から検索されます。	
3	空白にする必要があります。	
4	特定のキーワードを含むグローバル機能のみを検索するには、ここにそのキーワードを指定します。 キーワードを指定する場合、オプション（上記参照）としてキーワードタイプ（1、2、または 3）を指定する必要もあります。	

フィールド RESULT-FIELD (1:*n*) は、指定したリストを返します。

例：

入力：	出力：
Command Line 1: G Command Line 2: ADD	Result-Field 1: ADD CUSTOMER Result-Field 2: ADD FILE Result-Field 3: ADD USER

ローカル機能の HELP

このオプションは、指定したロケーションのすべてのローカル機能のリストを返します。

コマンド行	内容	
1	インジケータ "L" で始める必要があります。	
	オプション：	
	I	内部機能番号（IFN：Internal Function Number）も返されます。
	T	キーワードの一部を大文字で示します（可能な省略形を示すため）。
	S	RESULT-FIELD に返されたキーワードは列に整列されます。
	A	内部キーワードも返されます。
	1	タイプ 1 の特定のキーワードを含む機能のみが返されます。
	2	タイプ 2 の特定のキーワードを含む機能のみが返されます。
	3	タイプ 3 の特定のキーワードを含む機能のみが返されます。
	C	現在のロケーションに定義されている機能のみが返されます（コマンド行 3 は無視されます）。
	F	ローカル機能の「再帰的な」リストを呼び出します。つまり、現在のロケーション／指定したロケーションに関連するすべてのローカルコマンドが返されます。
2	ローカル機能検索の開始値（オプション）。 キーワードは "123" の順序で指定する必要があります。	
3	返されるリストに対応するロケーションを指定します。 キーワードは "123" の順序で指定する必要があります。 ロケーションを指定しなかった場合、コマンドプロセッサの現在のロケーションが使用されます。	
4	キーワードの制限（オプション）： キーワード、またはフォーマット N4 の IKN を指定した場合、このキーワードを含む機能のみが返されます。	

フィールド RESULT-FIELD (1:*n*) は、指定したリストを返します。

IKN の HELP

指定された内部キーワード番号（IKN）に対して、このオプションは元のキーワードを返します。

コマンド行	内容	
1	"IKN" で開始する必要があります。	
	オプション：	
	A	内部キーワードが示されます。
	T	キーワードの一部を大文字で示します（可能な省略形を示すため）。
2	変換する IKN をフォーマット N4 で指定します。	

フィールド `RESULT-FIELD (1)` は、キーワードを返します。

例：

入力：	出力：
`Command Line 1:  IKN` `Command Line 2:  0000002002`	`Result-Field 1:  CUSTOMER`

IFN の HELP

指定された内部機能番号（IFN）に対して、このオプションは機能のキーワードを返します。

コマンド行	内容	
1	"IFN" で開始する必要があります。	
	オプション：	
	A	内部キーワードを含む機能は省略されません。
2	変換する IFN をフォーマット N10 で指定します。	
3	**その他のオプション：**	
	S	IFN に属すキーワードが `RESULT-FIELD (1:3)` に返されます。
	T	キーワードの一部を大文字で示します（可能な省略形を示すため）。
	L	IFN をロケーションとして使用した場合、IFN が返されます。
	C	IFN をコマンドとして使用した場合、IFN が返されます。

フィールド `RESULT-FIELD(1)` は、機能を返します。オプション "S" を使用した場合、機能は `RESULT-FIELD (1:3)` に返されます。

例：

入力：	出力：
Command Line 1: IFN Command Line 2: 0001048578	Result-Field 1: DISPLAY INVOICE

全般的な情報の TEXT

全般的な情報については、COMMAND-LINE (*)、つまりすべてのコマンド行を空白にする必要があります。 RESULT-FIELD の最大 9 個のフィールドに次の情報が返されます。

RESULT-FIELD	内容	フォーマット
1	ユーザーテキスト用のヘッダー 1	テキスト（A40）
2	ユーザーテキスト用のヘッダー 2	テキスト（A40）
3	テキスト「First Entry used as」	テキスト（A16）
4	テキスト「Second Entry used as」	テキスト（A16）
5	テキスト「Third Entry used as」	テキスト（A16）
6	エントリ 1 のキーワード数	数値（N3）
7	エントリ 2 のキーワード数	数値（N3）
8	エントリ 3 のキーワード数	数値（N3）
9	カタログされた機能数	数値（N7）

キーワード情報の TEXT

キーワード情報については、COMMAND-LINE (1) には対応するキーワードが含まれている必要があります。COMMAND-LINE (2) にはオプションでキーワードタイプ（1、2、3、またはP）を含めることができます。COMMAND-LINE (3:6) は空にする必要があります。

RESULT-FIELD	内容	フォーマット
1	キーワードコメントテキスト	テキスト（A40）
2	完全な長さのキーワード	テキスト（A16）
3	一意の短縮形キーワード	テキスト（A16）
4	「Keyword used as」 エントリ	テキスト（A16）
5	内部キーワード番号（IKN）	数値（N4）
6	キーワードの最小長	数値（N2）
7	キーワードの最大長	数値（N2）
8	キーワードタイプ（1、2、3、1S、2S、3S、P）	テキスト（A2）

機能情報の TEXT

機能情報については、COMMAND-LINE (1:3) には目的のロケーションを指定するキーワードが含まれている必要があります。 COMMAND-LINE (4:6) には目的の機能を指定するキーワードが含まれている必要があります。 例えば、グローバルコマンド ADD USER に関する情報を返す場合、コマンド行 1、2、3、6 が空白であり、コマンド行 4 には "ADD"、コマンド行 5 には "USER" がそれぞれ含まれている必要があります。

RESULT-FIELD	内容	フォーマット
1	ランタイムアクション定義にオプション T で定義されたテキスト。	テキスト（A40）
2	指定したロケーションの内部機能番号（IFN）。	数値（N10）
3	指定した機能の内部機能番号（IFN）。	数値（N10）

GET オプション

GET オプションは、内部コマンドプロセッサ情報や現在のコマンドプロセッサ設定をダイナミックに割り当てられたバッファ NCPWORK から読み取るために使用します。 次のフィールドが使用されます。

フィールド名	内容
GETSET-FIELD-NAME (A32)	読み取る変数の名前。
GETSET-FIELD-VALUE (A32)	PROCESS COMMAND ACTION GET の実行後に指定した変数の値。

GETSET-FIELD-NAME に有効な値のリストについては、下記を参照してください。

SET オプション

SET オプションは、バッファ NCPWORK の内部コマンドプロセッサ設定を変更するために使用します。

フィールド名	内容
GETSET-FIELD-NAME (A32)	変更する変数の名前。
GETSET-FIELD-VALUE (A32)	指定した変数に書き込まれる値。

GETSET-FIELD-NAME の設定可能値：

フィールド名	フォーマット	G/S*	内容
NAME	A8	G	現在のプロセッサの名前。
LIBRARY	A8	G	ロード元のライブラリ。
FNR	N10	G	ロード元のファイル。
DBID	N10	G	ロード元のデータベース。
TIMESTMP	A8	G	現在のプロセッサのタイムスタンプ。
COUNTER	N10	G	アクセスカウンタ。
BUFFER-LENGTH	N10	G	NCPWORK 用に割り当てられたバイト数。
C-DELIMITER	A1	G/S	複数コマンドのデリミタ。
DATA-DELIMITER	A1	G	データの前に付けられるデリミタ。
PF-KEY	A1	G/S	PF キーをコマンドとして使用（Y/N）。
UPPER-CASE	A1	G	大文字のキーワード（Y/N）。
UQ-KEYWORDS	A1	G	一意のキーワード（Y/N）。
IMPLICIT-KEYWORD	A1	G/S	暗黙的なキーワードエントリの識別子。
MIN-LEN	N10	G	キーワードの最小長。
MAX-LEN	N10	G	キーワードの最大長。
KEYWORD-SEQ	A8	G/S	キーワードシーケンス。
ALT-KEYWORD-SEQ	A8	G/S	代替キーワードシーケンス。
USER-SEQUENCE	A1	G	ユーザーによる KEYWORD-SEQ の上書きを許可（Y/N）。
CURR-LOCATION	N10	G/S	現在のロケーション（IFN）。
CURR-IKN1	N10	G/S	現在のロケーションの IKN1。
CURR-IKN2	N10	G/S	現在のロケーションの IKN2。
CURR-IKN3	N10	G/S	現在のロケーションの IKN3。
CHECK-LOCATION	N10	G	最後にチェックされたロケーション（IFN）。
CHECK-IKN1	N10	G	CHECK-LOCATION の IKN1。
CHECK-IKN2	N10	G	CHECK-LOCATION の IKN2。
CHECK-IKN3	N10	G	CHECK-LOCATION の IKN3。
TOP-IKN1	N10	G	先頭キーワードの IKN1。
TOP-IKN2	N10	G	先頭キーワードの IKN2。
TOP-IKN3	N10	G	先頭キーワードの IKN3。
KEY1-TOTAL	N10	G	タイプ 1 のキーワードの数。
KEY2-TOTAL	N10	G	タイプ 2 のキーワードの数。
KEY3-TOTAL	N10	G	タイプ 3 のキーワードの数。
FUNCTIONS-TOTAL	N10	G	カタログされた機能の数。
LOCAL-GLOBAL-SEQ	A8	G/S	ローカル／グローバル機能の確認。

フィールド名	フォーマット	G/S*	内容
ERROR-HANDLER	A8	G/S	一般的なエラープログラム。
SECURITY	A1	G	Natural Security がインストール済み（Y/N）。
SEC-PREFETCH	A1	G	Natural Security データを読み取る（Y/N）または読み取り済み（D = 終了）
PREFIX1	A1	G	プロセッサヘッダーメンテナンス2画面のフィールド Prefix Character 1 に対応。
PREFIX2	A1	G	プロセッサヘッダーメンテナンス2画面のフィールド Prefix Character 2 に対応。
HEX1	A1	G	プロセッサヘッダーメンテナンス 2 画面のフィールド Hex. Replacement 1 に対応。
HEX2	A1	G	プロセッサヘッダーメンテナンス 2 画面のフィールド Hex. Replacement 2 に対応。
DYNAMIC	A32	G	最後のエラーメッセージのダイナミック部分（:*n*:）。
LAST	-	G	データとしてスタックの先頭に置かれる最後のコマンド。
LAST-ALL	-	G	データとしてスタックの先頭に置かれる最後のコマンド群。
LAST-COM	-	G	*COM に移される最後のコマンド。
MULTI	-	G	最後の複数コマンドをデータとしてスタックの先頭に挿入。
MULTI-COM	-	G	最後の複数コマンドをシステム変数 *COM に挿入。

*G = GET オプションで使用できます。

*S = SET オプションで使用できます。

USING 節

フィールドの内容（プロセッサ名やコマンド行など）を USING 節に指定します。

USING 節では、コマンドプロセッサに送信するフィールドを指定します。

オプション	フィールド名			
	PROCESSOR-NAME	COMMAND-LINE	GETSET-FIELD-NAME	GETSET-FIELD-VALUE
CLOSE				
CHECK	必須	必須		
EXEC	必須	必須		
TEXT	必須	必須		
HELP	必須	必須		
GET	必須		必須	
SET	必須		必須	必須

GIVING 節

Note: この節はレポーティングモードでのみ使用できます。

GIVING節では、各オプションの処理結果として、コマンドプロセッサによって埋められるフィールドを指定します。

オプション	フィールド名			
	NATURAL-ERROR	RETURN-CODE	RESULT-FIELD	GETSET-FIELD-VALUE
CLOSE	推奨			
CHECK	推奨	必須	必須	
EXEC	推奨	必須	必須	
TEXT	推奨	必須	必須	
HELP	推奨	必須	必須	
GET	推奨			必須
SET	推奨			

Note: GIVING節はストラクチャードモードでは利用できません。DEFINE DATAステートメントに指定されたすべてのフィールドで構成される暗黙的なGIVING節が存在するためです。通常、これらのフィールドはレポーティングモードのGIVING節で参照されます。つまり、ストラクチャードモードでは、上記の表の「必須」でマークされたフィールドはすべてDEFINE DATAステートメントで定義する必要があります。

DDM：COMMAND

データ定義モジュール（DDM）"COMMAND" は、PROCESS COMMANDステートメントと組み合わせて使用するために特別に作成されたものです。

```
DB:    1 File:    1 - COMMAND                          Default Sequence: ?

 TYL  DB  NAME                            F LENG  S D   REMARKS
 ---  --  ------------------------------- - ----  - -  ------------------------
   1  AA  PROCESSOR-NAME                  A    8  N D DE    USING
 M 1  AB  COMMAND-LINE                    A   80  N D MU/DE USING
   1  AF  GETSET-FIELD-NAME               A   32  N D DE    USING
   1  BA  NATURAL-ERROR                   N  4.0  N         GIVING
   1  BB  RETURN-CODE                     A    4  N         GIVING
 M 1  BC  RESULT-FIELD                    A   80  N   MU    GIVING
   1  BD  GETSET-FIELD-VALUE              A   32  N D       USING; GIVING
```

```
***** DDM OUTPUT TERMINATED ******
```

Note: コンパイルエラーやランタイムエラーを回避するため、使用前にDDM"COMMAND"がタイプ"C"（SYSDDMメニューのDDM Typeフィールド）でカタログされていることを確認してください。DDMを再カタログする場合、SYSDDMユーティリティのDBID/FNR指定は無視されます。

DDM "COMMAND" のフィールドは次のとおりです。

DDMフィールド	説明
PROCESSOR-NAME	PROCESS COMMANDステートメントが発行されるコマンドプロセッサの名前です。 指定するコマンドプロセッサはカタログする必要があります。
COMMAND-LINE	コマンドプロセッサで処理されるコマンド行（オプションCHECK、EXEC）、またはプログラムに返されるユーザーテキストやヘルプテキストに対するキーワード／コマンド（オプションTEXT、HELP）です。 このフィールドは複数行指定できます。
GETSET-FIELD-NAME	GETおよびSETオプションとともに使用し、読み取り（GET）または書き込み（SET）が行う定数や変数の名前を指定します。
RETURN-CODE	このフィールドには、ランタイムアクション定義内で指定されたオプションEXECやCHECKによる処理のリターンコードが入ります（Natural SYSNCPユーティリティを参照）。
NATURAL-ERROR	すべてのオプションと組み合わせて使用します。 DEFINE DATAでこのフィールドを使用した場合、コマンドプロセッサに対するNaturalエラーコードが返されます。 フィールドが存在しない場合、エラーの発生時にNaturalランタイムエラー処理が開始されます。
RESULT-FIELD	このフィールドには、ランタイムアクション定義内で指定された各種オプションの使用結果情報が入ります（Natural SYSNCPユーティリティのランタイムアクションを参照）。 このフィールドは複数行指定できます。
GETSET-FIELD-VALUE	GETおよびSETオプションとともに使用し、GETSET-FIELD-NAMEフィールドで指定された定数や変数の値が入ります。

例

- 例 1 - PROCESS COMMAND ACTION CLOSE

例 1 - PROCESS COMMAND ACTION CLOSE

```
/* EXAM-CLS - Example for PROCESS COMMAND ACTION CLOSE (Structured Mode)
/************************************************************************
DEFINE DATA LOCAL
  01 COMMAND VIEW OF COMMAND
END-DEFINE
/*
PROCESS COMMAND ACTION CLOSE
/*
DEFINE WINDOW CLS
INPUT WINDOW = 'CLS'
  'NCPWORK has just been released.'
/*
END
```

例 - PROCESS COMMAND ACTION EXEC2

```
/* EXAM-EXS - Example for PROCESS COMMAND ACTION EXEC (Structured Mode)
/************************************************************************
DEFINE DATA LOCAL
  01 COMMAND VIEW OF COMMAND
     02 PROCESSOR-NAME
     02 COMMAND-LINE (1)
     02 NATURAL-ERROR
     02 RETURN-CODE
     02 RESULT-FIELD (1)
  01 MSG (A65) INIT <'Please enter a command.'>
END-DEFINE
/*
REPEAT
  INPUT (AD=MIT' ' IP=OFF) WITH TEXT MSG
    'Example for PROCESS COMMAND ACTION EXEC (Structured Mode)' (I)
  / 'Command ==>' COMMAND-LINE (1) (AL=64)
  /*******
 PROCESS COMMAND ACTION EXEC
    USING
      PROCESSOR-NAME = 'DEMO'
      COMMAND-LINE (1) = COMMAND-LINE (1)
  /*******
  COMPRESS 'NATURAL-ERROR =' NATURAL-ERROR TO MSG
END-REPEAT
END
```

Note: 他のプログラム例については、SYSNCP ライブラリを参照してください。 これらのプログラムはすべて "EXAM" で始まります。

99 PROCESS GUI

```
PROCESS GUI ACTION action-name WITH { { operand1 }...  } [GIVING operand2]
                                    { { nX       }     }
                                    { PARAMETERS-clause }
```

このchapterでは、次のトピックについて説明します。

関連ステートメント：**OPEN DIALOG** | **CLOSE DIALOG** | **SEND EVENT**

関連機能グループ：「*イベントドリブンプログラミング*」

機能

PROCESS GUI ステートメントは、アクションを実行するために使用します。 ここでのアクションとは、イベントドリブンアプリケーションで頻繁に必要とされるプロシージャを指します。

これらの標準プロシージャの全般的な情報については、「*イベントドリブンプログラミング手法*」（『*プログラミングガイド*』）を参照してください。

有効な個々のアクション、そのパラメータ、および例については、「*PROCESS GUI ステートメントアクション*」（『*ダイアログコンポーネントリファレンス*』）を参照してください。

構文説明

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
*operand1**	C	S	A			A	U	N	P	I	F	B	D	T	L		G	可	不可
operand2		S						N	P	I								可	不可

* 実際に可能な構造およびフォーマットは実行するアクションによって変わります。

action-name	呼び出すアクション： *action-name* として、呼び出されるアクションを指定します。
operand1	アクションへのパラメータの引き渡し： *operand1* として、アクションに渡されるパラメータを指定します。パラメータは、指定した順序で渡されます。
PARAMETERS	下記の「*名前によるパラメータの引き渡し*」を参照してください。
***n*X**	省略されるパラメータ：

	表記 *nX* を使用して、次の *n* パラメータを省略するように指定できます。例えば、1X は次のパラメータを省略し、3X は次の 3 つのパラメータを省略します。これは、次の *n* パラメータに、アクションに渡す値がないことを意味します。 これは、ActiveX コントロールに適用されるアクションに対してのみ有効です。 省略されるパラメータは、ActiveX コントロールのメソッドで 「オプション」 として定義する必要があります。 パラメータを 「オプション」 として定義すると、呼び出し側オブジェクトからこれらのパラメータに値を（必要なくても）渡すことができます。
GIVING *operand2*	レスポンスコード用のフィールド： *operand2* として、呼び出されたアクションの実行後にアクションからのレスポンスコードを受け取るフィールドを指定できます。

名前によるパラメータの引き渡し：

アクション「ADD」の場合は、位置の代わりに名前でパラメータを渡すこともできます。そのためには、*PARAMETERS-clause* を使用します。

```
PARAMETERS {parameter-name=operand1} ...
END-PARAMETERS
```

この節は、アクション 「ADD」 にのみ使用でき、他のアクションには使用できません。

アクションにオプションのパラメータ（指定する必要のないパラメータ）がある場合、表記 *nX* を使用できます。*n* は指定しないパラメータの代わりとなります。現在、オプションのパラメータを持つことのできるアクションは、ActiveX コントロールのメソッドとパラメータ化されるプロパティだけです。

100 PROCESS PAGE

PROCESS PAGE ステートメントは、外部レンダリングエンジン（Application Designer など）に一般的なインターフェイスの説明を提供し、Natural内部データ表現と外部データ表現をリンクします。 このリンクを介して、データおよびイベントがブラウザベースの外部アプリケーションとやりとりされます。ただし、レンダリング情報はやりとりされません。

詳細については、『*Natural for Ajax*』ドキュメントを参照してください。

このchapterでは、次のトピックについて説明します。

構文 1 - PROCESS PAGE

```
PROCESS PAGE operand1
 [WITH PARAMETERS
  {[NAME] operand2 [VALUE] operand3 [ { (EMU=emvalue) } ]}...
                                     { (EM=emvalue)  }
 END-PARAMETERS]
 [GIVING operand9]
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連機能グループ：「*対話型処理用の画面生成*」

構文説明 - 構文 1

PROCESS PAGE ステートメントの構文 1 は、通常 Natural アダプタ内でのみ使用されます。 アダプタとは、Natural アプリケーションと Web ページ間のインターフェイスを形成する Natural オブジェクトです。 アダプタは、レイアウトの保存時に Application Designer によって自動的に作成／更新されます。

Note:

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1	C	S				A	U												可	不可
operand2	C	S				A	U												可	不可
operand3	C	S	A			A	U	N	P	I	F	B	D	T	L				可	可
operand9		S								I4									可	可

構文要素の説明：

operand1	外部ページレイアウトの名前です。
operand2	*operand3* の相互転送に使用される外部データフィールドの名前が入ります。
operand3	転送される Natural データフィールドの名前が入ります。
GIVING *operand9*	**GIVING** 節： 要求を実行できなかった場合は、*operand9* に Natural エラーが含まれます。
EMU= **EM=**	データ転送時に使用される編集マスク。 詳細については、『パラメータリファレンス』のセッションパラメータ EM を参照してください。 Unicode 編集マスクの詳細については、『パラメータリファレンス』でセッションパラメータ EMU を参照してください。

Application Designer によって作成されたアダプタの例：

```
* PAGE1: PROTOTYPE       --- CREATED BY Application Designer ---
* PROCESS PAGE USING 'XXXXXXXX' WITH
* INFOPAGENAME RESULT YOURNAME
DEFINE DATA PARAMETER
1 INFOPAGENAME (U) DYNAMIC
1 RESULT (U) DYNAMIC
1 YOURNAME (U) DYNAMIC
END-DEFINE
*
PROCESS PAGE U'/njxdemos/helloworld' WITH
PARAMETERS
 NAME U'infopagename'
  VALUE INFOPAGENAME
 NAME U'result'
  VALUE RESULT
 NAME U'yourname'
  VALUE YOURNAME
END-PARAMETERS
*
*  TODO: Copy to your calling program and implement.
/*/*( DEFINE EVENT HANDLER
* DECIDE ON FIRST *PAGE-EVENT
*  VALUE U'nat:page.end'
*   /* Page closed.
*   IGNORE
*  VALUE U'onHelloWorld'
*   /* TODO: Implement event code.
*   PROCESS PAGE UPDATE FULL
```

```
*  NONE VALUE
*   /* Unhandled events.
*   PROCESS PAGE UPDATE
* END-DECIDE
/*/*) END-HANDLER
*
END
```

構文 2 - PROCESS PAGE USING

```
PROCESS PAGE USING operand4
    [ {     WITH {operand5} ... }     ]
     {     NO PARAMETER          }
 [GIVING operand9]
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連機能グループ：「*対話型処理用の画面生成*」

構文説明 - 構文 2

この構文は、ページレイアウト（Application Designer または類似のツールで作成）から生成されたアダプタタイプのオブジェクトを使用してリッチ GUI 入力／出力処理を実行するために使用します。

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand4	C	S				A													可	不可
operand5		S	A	G		A	U	N	P	I	F	B	D	T	L				可	可
operand9		S								I4									可	可

構文要素の説明：

USING *operand4*	アダプタ名： Natural システムファイルに事前に保存されているアダプタ定義を呼び出します。『*プログラミングガイド*』の「*リッチ GUI ページの処理 - アダプタ*」も参照してください。 アダプタ名（*operand4*）には、1～8 文字の英数字定数またはユーザー定義変数を使用できます。 変数を使用する場合、事前の定義が必要です。

	アダプタ名にはアンパサンド（&）を含めることができます。この文字は実行時に Natural システム変数 *LANGUAGE の現在の値で置き換えられます。 この機能は履歴上の理由で用意されています。 複数言語のアダプタが必要な場合、外部レンダリングシステム（例えば、Application Designer）の機能を使用してください。 **注意:** 新しいアプリケーションでは、複数言語にするためのアンパサンド機能は必要ありません。 Application Designer などを使用して設計されたページでは、レイアウト設計の一部として複数言語の情報を保持できます。 『*Natural for Ajax*』ドキュメントの「*Multi Language Management*」を参照してください。
operand5	フィールド指定： データベースフィールドまたはユーザー定義変数（すべて事前の定義が必要）のリストを指定します。フィールドは、番号、シーケンス、フォーマット、長さ、オカレンス数（配列の場合）について、参照されているアダプタに準拠している必要があります。そうでない場合、エラーが発生します。 データベースフィールドの内容が PROCESS PAGE 処理の結果として変更される場合は、データエリアにある値のみが変更されます。 データベースの内容を変更するためには、適切なデータベース UPDATE/STORE ステートメントを使用する必要があります。 「*プログラムで定義されたフィールドに基づく **PROCESS PAGE USING***」を参照してください。
NO PARAMETER	「*パラメータリストなしの **PROCESS PAGE USING***」を参照してください。
GIVING *operand9*	**GIVING** 節： 要求を実行できなかった場合は、*operand9* に Natural エラーが含まれます。 **注意:** GIVING 節では、アダプタオブジェクトの起動中または実行中にエラーが発生した場合に共通 Natural エラー処理を中断します。 Natural モジュールを後方追跡して ON ERROR 節を見つける代わりに、Natural エラーコードが変数（*operand9*）に渡され、次のステートメントの実行が続行されます。

パラメータリストなしの PROCESS PAGE USING

PROCESS PAGE USING をパラメータリストなしで使用する場合、次の要件を満たす必要があります。

- アダプタ名（*operand5*）は、英数字定数（最大 8 文字）として指定する必要があります。
- この方法で使用されるアダプタは、そのアダプタを参照するプログラムのコンパイル前に事前に作成されている必要があります。
- 処理対象のフィールドの名前は、コンパイル時にアダプタソース定義からダイナミックに取得される必要があります。 プログラムとアダプタの両方で使用されるフィールド名が同一である必要があります。
- PROCESS PAGE ステートメントで参照されるすべてのフィールドにその時点でアクセスできる必要があります。

- ストラクチャードモードでは、フィールドが事前に定義されている必要があります（処理ループまたはビューでデータベースフィールドが適切に参照されている必要があります）。
- ページレイアウトが変更されたとき、アダプタを使用しているプログラムの再カタログは不要です。 ただし、配列構造や名前、フィールドのフォーマット／長さが変更されたとき、またはアダプタでフィールドが追加／削除されたときは、アダプタを使用しているプログラムの再カタログが必要になります。
- プログラムのコンパイル時にアダプタソースを利用できる必要があります。利用できない場合、`PROCESS PAGE USING` ステートメントはコンパイルできません。

Note: アダプタを利用できない場合でもプログラムをコンパイルする必要がある場合は、`NO PARAMETER` を指定してください。 これにより、アダプタを利用できない場合でも `PROCESS PAGE USING` ステートメントのコンパイルが可能になります。

プログラムで定義されたフィールドに基づく **PROCESS PAGE USING**

プログラム（*operand5*）内で処理するフィールドの名前を指定することで、プログラム内のフィールドとアダプタ内のフィールドに異なる名前を使用できます。

プログラム内のフィールドのシーケンスは、アダプタ内のフィールドのシーケンスと一致している必要があります。 Natural マップをアダプタオブジェクトとして使用する場合は、マップエディタによって、マップで指定されたフィールドがフィールド名でアルファベット順にソートされることに留意してください。 詳細については、『エディタ』ドキュメントのマップエディタの説明を参照してください。

実行時に、プログラムで指定されたフィールドのフォーマットと長さがアダプタで指定されたフィールドと一致しているかどうかがチェックされます。 両方のレイアウトが一致しない場合は、エラーメッセージが生成されます。

構文 3 - PROCESS PAGE UPDATE

```
PROCESS PAGE UPDATE [FULL] [event-option]
 [GIVING operand9]
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連機能グループ：「*対話型処理用の画面生成*」

構文説明 - 構文 3

PROCESS PAGE UPDATE ステートメントは、PROCESS PAGE ステートメントに戻って再実行するために使用します。通常、前の PROCESS PAGE ステートメントのデータ入力処理が不完全だったイベント処理から戻るために使用します。

Note: PROCESS PAGE ステートメントとそれに対応する PROCESS PAGE UPDATE ステートメントの間に、INPUT、WRITE、PRINT、または DISPLAY ステートメントを実行することはできません。

PROCESS PAGE UPDATE ステートメントが実行されると、PROCESS PAGE ステートメントが実行されたときに存在していたサブルーチン、特殊条件、およびループ処理を考慮してプログラムステータスが再位置づけされます（PROCESS PAGE ステートメントのステータスがまだ有効な場合）。 PROCESS PAGE ステートメントの実行後にループが開始され、PROCESS PAGE UPDATE ステートメントがそのループ内にある場合、ループは中断され、PROCESS PAGE UPDATE の結果として再処理された PROCESS PAGE ステートメントの後で再開されます。

PROCESS PAGE ステートメントの実行後にサブルーチン階層が呼び出され、サブルーチン内で PROCESS PAGE UPDATE が実行された場合、Natural は自動的に全サブルーチンをさかのぼり、PROCESS PAGE ステートメントの位置にプログラムステータスを再位置づけします。

PROCESS PAGE ステートメントが実行されたステータスがすでに終了してしまった時点では、PROCESS PAGE ステートメントをループ、サブルーチン、または特殊条件ブロック内に位置づけ、PROCESS PAGE UPDATE ステートメントを実行することはできません。 このエラー条件が検出されると、エラーメッセージが生成され、プログラムの実行が終了します。

オペランド定義テーブル：

オペランド	構文要素	フォーマット	ステートメント参照	ダイナミック定義
operand9	S	I4	可	可

構文要素の説明：

FULL	PROCESS PAGE UPDATE ステートメントに FULL オプションを指定すると、対応する PROCESS PAGE ステートメントが完全に再実行されます。 ■ 通常の PROCESS PAGE UPDATE ステートメント（FULL オプションなし）では、PROCESS PAGE ステートメントと PROCESS PAGE UPDATE ステートメントの間で変更された変数の内容は表示されません。つまり、画面上の変数の内容はすべて PROCESS PAGE ステートメントが実行された時点のものになります。 ■ PROCESS PAGE UPDATE FULL ステートメントでは、最初の PROCESS PAGE ステートメントの実行後に行われたすべての変更は、PROCESS PAGE ステートメントが再実行されたときに適用されます。つまり、画面上の変数の値はすべて PROCESS PAGE UPDATE ステートメントが実行された時点のものになります。

event-option	**EVENT** オプション： 下記の「*EVENT オプション*」を参照してください。
GIVING (*operand9*)	**GIVING** 節： 要求を実行できなかった場合は、*operand9* に Natural エラーが含まれます。

ユーザープログラムフラグメントの例：

```
PROCESS PAGE USING "HELLOW-A"
*
/*( DEFINE EVENT HANDLER
DECIDE ON FIRST *PAGE-EVENT
 VALUE U'nat:page.end'
  /* Page closed.
  IGNORE
 VALUE U'onHelloWorld'
  COMPRESS "HELLO WORLD" YOURNAME INTO RESULT
  PROCESS PAGE UPDATE FULL
 NONE VALUE
  /* Unhandled events.
  PROCESS PAGE UPDATE
END-DECIDE
/*) END-HANDLER
```

EVENT オプション

```
AND SEND EVENT operand6
 [WITH PARAMETERS
  {[NAME] operand7 [VALUE] operand8 [ { (EMU=value) } ]}...
                                     { (EM=value)  }
 END-PARAMETERS]
```

このオプションでは、外部 I/O システムに特定の機能を実行するように指示できます。 これらの機能は、外部I/Oシステムの一部であるか、またはフォーカスの設定、メッセージボックスの表示などの出力処理に関する特別な機能を実装します。

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand6	C	S				A	U												可	不可
operand7	C	S				A	U												可	不可
operand8	C	S	A			A	U	N	P	I	F	B	D	T	L				可	可

構文要素の説明：

operand6	外部 **I/O** システムから要求されるイベント： 外部 I/O システムの実装に応じて、イベントを使用できます。Application Designer ドキュメントを参照してください。
operand7	外部データフィールド名： *operand7* には、*operand8* の相互転送に使用される外部データフィールドの名前が入ります。
operand8	**Natural** データフィールド： *operand8* には、転送される Natural データフィールドが入ります。
EMU= **EM=**	編集マスク： データ転送時に使用される編集マスク。 編集マスクの詳細については、『パラメータリファレンス』でセッションパラメータ EM を参照してください。 Unicode 編集マスクの詳細については、『パラメータリファレンス』でセッションパラメータ EMU を参照してください。

構文 4 - PROCESS PAGE MODAL

```
PROCESS PAGE MODAL
  statement ...
END-PROCESS
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ESCAPE** | **PROCESS PAGE**

関連機能グループ：

- ループ実行

■ 対話型処理用の画面生成

構文説明 - 構文 4

PROCESS PAGE MODAL ステートメントは、処理ブロックを開始し、モーダルリッチ GUI ウィンドウの存続期間を制御するために使用します。

Note: PROCESS PAGE MODAL ステートメントはバッチモードでは使用できません。

PROCESS PAGE MODAL ステートメントブロックに入ると、まだ表示されていないレポート 0 のデータが最初に表示されます。

システム変数 *PAGE-LEVEL が増加し、モーダルページのオープン準備が行われます。 モーダルページの実際のオープンは次の PROCESS PAGE USING '*adapter*' WITH ステートメントで実行されます。

Note: PROCESS PAGE MODAL ステートメントとそれに対応する END-PROCESS ステートメントの間に、レポート 0 を参照する PRINT、WRITE、INPUT、または DISPLAY ステートメントを実行することはできません。

PROCESS PAGE MODAL ステートメントブロックを抜けると、次のアクションが実行されます。

- モーダルページがこのレベルでオープンされている場合、モーダルページがクローズされます。
- システム変数 *PAGE-LEVEL が減少し、システム変数 *PAGE-EVENT が、ステートメントブロックに入る前の値に戻されます。

構文要素の説明：

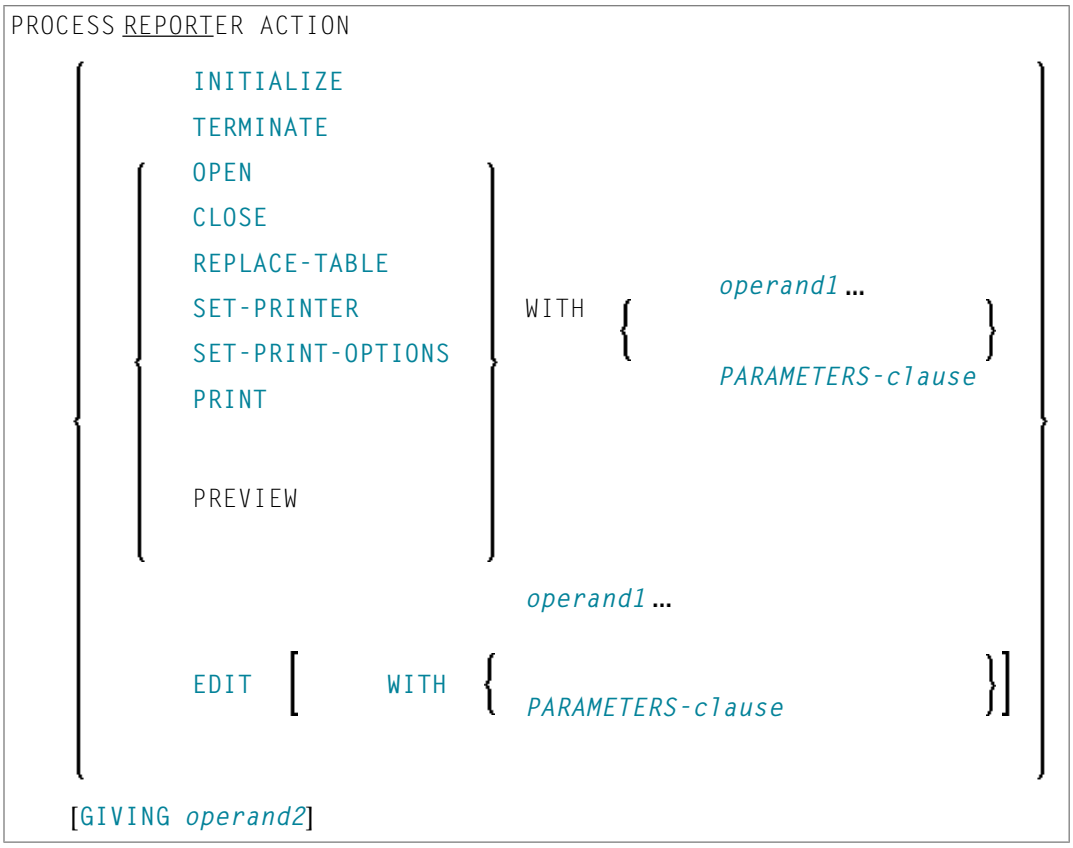
statement	状況に応じて、*statement* の代わりに、1 つ以上の適切なステートメントを指定する必要があります。不要なステートメントがある場合は、IGNORE ステートメントを挿入します。
END-PROCESS	PROCESS PAGE MODAL ステートメントを終了するには、Natural 予約語 END-PROCESS を使用する必要があります。

例：

```
* Name: First Demo/Open modal!
*
PROCESS PAGE USING "EMPTY-A"
*
/*( DEFINE EVENT HANDLER
DECIDE ON FIRST *PAGE-EVENT
  VALUE U'nat:page.end', U'onClose'
    /* Page closed.
    IGNORE
```

```
  VALUE U'onNextLevel'
    PROCESS PAGE MODAL
      FETCH RETURN "EMPTY-P"
    END-PROCESS
    PROCESS PAGE UPDATE
  NONE VALUE
    PROCESS PAGE UPDATE
END-DECIDE
/*) END-HANDLER
END
```

101 PROCESS REPORTER



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

PROCESS REPORTER ステートメントは、プログラム内から Natural Reporter と通信し、特定のアクションを実行するように Reporter に指示するために使用します。

Reporter については、Natural Reporter オンラインヘルプを参照してください。

Note: 特定のレポートに適用するアクションについて、2 番目のキーワードは REPORT に短縮できます。 これはプログラムを読みやすくするためだけのものです。キーワードを短縮してもしなくても Natural では区別されません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A		N	P	I	F	B	D	T	L				可	不可
operand2		S						N	P	I									可	不可

構文要素の説明：

ACTION	アクション： 次のアクションのいずれかを指定して Reporter に実行させることができます。
INITIALIZE	Reporter を初期化し、ロードします。 常に、実行する最初のアクションにする必要があります。
TERMINATE	Reporter を終了し、アンロードします。 常に、実行する最後のアクションにする必要があります。
OPEN	指定したレポートをオープンし、後続のアクションに対してレポートを識別するために使用できるハンドルを返します。
CLOSE	レポートハンドルが使用できなくなった後、指定したレポートをクローズします。
REPLACE-TABLE	テーブルのパス名を置き換えます。
SET-PRINTER	後続のすべてのレポート出力に使用するプリンタを選択します。 選択されたプリンタに対する出力方法は NATPARM で "TTY" に設定する必要があります。
SET-PRINT-OPTIONS	指定したレポートの出力オプションを設定します。
PRINT	指定したレポートを現在選択されているプリンタに出力します。
PREVIEW	指定したレポートを現在選択されているプリンタを基準にプレビューします。
EDIT	レポートを指定していない場合は、Reporter のメインウィンドウを表示します。レポートを指定した場合は、そのレポートに対する編集ウィンドウとともに Reporter のメインウィンドウを表示します。
WITH	**WITH** 節： *operand1* として、アクションに渡されるパラメータを指定します。
PARAMETERS-clause	WITH 節の代わりに、下記の *PARAMETERS-clause*を使用することもできます。
GIVING *operand2*	**GIVING** 節： GIVING節では、呼び出したアクションからのレスポンスコードを受け取ることができます。 *operand2* として、レスポンスコードを受け取るフィールドを指定します。 レスポンスコードはフォーマット／長さ I4 で返されます。

	レスポンスコード "0" は、アクションが正常に行われたことを示します。 他のレスポンスコードはすべて Natural システムエラー番号（NAT*nnnn*）に対応します。

PARAMETERS 節

```
PARAMETERS {parameter-name=operand1} ...
END-PARAMETERS
```

この節では、パラメータ（複数可）を名前（位置の代わり）で指定します。

- OPEN アクションに対するパラメータ
- REPLACE-TABLE アクションに対するパラメータ
- SET-PRINTER アクションに対するパラメータ
- SET-PRINT-OPTIONS アクションに対するパラメータ
- CLOSE、PRINT、PREVIEW、EDIT アクションに対するパラメータ

OPEN アクションに対するパラメータ

このアクションに対しては、最初のパラメータとしてオープンするレポートの名前（*.rpt* 拡張子またはパス指定なし）を指定し、2番目のパラメータとしてハンドルを受け取るフィールドを指定します。 最初のパラメータのフォーマット／長さは A8 との互換性が必要で、2 番目のパラメータについては I4 との互換性が必要です。

レポートは、最初にログオンライブラリの RES サブディレクトリで検索され、次に各 STEPLIB の RES サブディレクトリで検索され、その後、環境変数 `NATGUI_BMP` に割り当てられたディレクトリで検索されます。

レポートデータは、最初に、レポートが作成されたときに指定したパス（存在する場合）で検索され、次にレポートが見つかったディレクトリで検索されることに注意してください。

*PARAMETERS-clause*を使用する場合、*parameter-name*は次のものにする必要があります。

- `REPORT-NAME`（レポート名）
- `REPORT-ID`（ハンドルフィールド）

「*例 1 - OPEN アクションに対するパラメータ*」も参照してください。

REPLACE-TABLE アクションに対するパラメータ

このアクションに対しては、最初のパラメータとしてアクションが適用されるレポートを識別するハンドルを指定し、2番目のパラメータとしてワークファイル番号を指定し、3番目のパラメータ（オプション）としてテーブル名を指定します。 最初の 2 つのパラメータのフォーマット／長さは I4 との互換性が必要で、3 番目のパラメータについては A8 との互換性が必要です。

*PARAMETERS-clause*を使用する場合、*parameter-names*はそれぞれ REPORT-ID、WORK-FILE、および TABLE-NAME にする必要があります。

「*例 2 - REPLACE-TABLE アクションに対するパラメータ*」も参照してください。

SET-PRINTER アクションに対するパラメータ

このアクションに対しては、*operand1* として、選択するプリンタの論理デバイス名（LPT1～LPT31）を指定します。 *operand1* のフォーマット／長さは A8 との互換性が必要です。

*PARAMETERS-clause*を使用する場合、*parameter-name*は DEVICE-NAME にする必要があります。

「*例 3 - SET-PRINTER アクションに対するパラメータ*」も参照してください。

SET-PRINT-OPTIONS アクションに対するパラメータ

このアクションに対しては、最初のパラメータ（下記の表の No. 1）としてアクションが適用されるレポートを識別するハンドルを指定し、その後に各種プリンタオプション（すべて任意）を指定します。 パラメータを省略すると、対応するオプションは変更されません。

シーケンス番号	パラメータ
1	このパラメータ（フォーマット／長さは I4 と互換性が必要）は、アクションが適用されるレポートを識別するハンドルです。 *parameter-name*は REPORT-ID にする必要があります。 *
2	このパラメータ（フォーマット／長さは I2 と互換性が必要）は、ローカルデータエリア NGULKEY1 に定義された用紙サイズ定数の 1 つです。 ここで可能な値： ■ CUSTOM-PAPER（用紙の縦と横を明示） ■ LETTER（8.5×11 インチ） ■ LEGAL（8.5×14 インチ） ■ EXECUTIVE（7.25×10.5 インチ） ■ A4（210×297 mm） ■ COM-10-ENVELOPE（4.125×9.5 インチ） ■ DL-ENVELOPE（110×220 mm） ■ C5-ENVELOPE（162×229 mm）

シーケンス番号	パラメータ
	■ B5-ENVELOPE（176×250 mm） ■ MONARCH-ENVELOPE（3.875×7.5 インチ） *parameter-name*は PAPER-SIZE にする必要があります。*
3 および 4	これらのパラメータ（フォーマット／長さは I2 と互換性が必要）は、それぞれ用紙の横と縦です（twip 単位：1 twip = 1/1440 インチ）。 これらのパラメータは、用紙サイズが CUSTOM-PAPER のときにのみ使用できます。 *PARAMETERS-clause*を使用する場合、*parameter-names*はそれぞれ PAPER-WIDTH および PAPER-HEIGHT にする必要があります。
5、6、7、および 8	これらのパラメータ（フォーマット／長さは I2 と互換性が必要）は、それぞれ、左、上、右、下の余白（twip 単位）です。 *parameter-names*はそれぞれ LEFT-MARGIN、TOP-MARGIN、RIGHT-MARGIN、BOTTOM-MARGIN にする必要があります。
9	このパラメータ（フォーマットは必ず L）は、用紙の向きです。 ■ TRUE = 横置き ■ FALSE = 縦置き このパラメータは、用紙サイズが CUSTOM-PAPER のときは使用できません。 *parameter-name*は LANDSCAPE にする必要があります。*
10	このパラメータ（フォーマットは必ずL）は、高速（テキストのみ）出力オプションです。 ■ TRUE = グラフィックの省略 ■ FALSE = 省略なし *parameter-name*は FAST-PRINT にする必要があります。*
11	このパラメータ（フォーマットは必ずL）では、全体が空白のレコードの出力を省略するかどうかを指定します。 ■ TRUE = 省略 ■ FALSE = 省略なし *parameter-name*は SUPPRESS- BLANK-LINES にする必要があります。*
12	このパラメータ（フォーマットは必ずL）では、同じデータを持つ連続するレコードを無視するかどうかを指定します。 ■ TRUE = 無視 ■ FALSE = 無視しない *parameter-name*は IGNORE-DUPLICATES にする必要があります。*
13	このパラメータ（フォーマットは必ずL）は、出力時にプリンタ選択ダイアログを表示するかどうかを指定します。

シーケンス番号	パラメータ
	■ TRUE = 表示 ■ FALSE = 表示しない *parameter-name*は SHOW-PRINT-DIALOG にする必要があります。 *
14	このパラメータ（フォーマット／長さは I2 と互換性が必要）は、ローカルデータエリア NGULKEY1 に定義された用紙ソース定数の 1 つです。 ここで可能な値： ■ AUTOMATIC = 自動 ■ MANUAL = 手差し *parameter-name*は PAPER-SOURCE にする必要があります。 *

Note: * *PARAMETERS-clause*を使用する場合に適用されます。

「*例 4 - SET-PRINT-OPTIONS アクションに対するパラメータ*」も参照してください。

CLOSE、PRINT、PREVIEW、EDIT アクションに対するパラメータ

これらのアクションに対しては、*operand1* として、アクションが適用されるレポートを識別するハンドルを指定します。 *operand1* のフォーマット／長さは I4 との互換性が必要です。

*PARAMETERS-clause*を使用する場合、*parameter-name*は REPORT-ID にする必要があります。

「*例 5 - CLOSE、PRINT、PREVIEW、EDIT アクションに対するパラメータ*」も参照してください。

例

例 1 - OPEN アクションに対するパラメータ

```
PROCESS REPORT ACTION OPEN WITH 'MYREPORT' #HANDLE
```

```
PROCESS REPORT ACTION OPEN WITH
  PARAMETERS
    REPORT-NAME = 'MYREPORT'
    REPORT-ID   = #HANDLE
  END-PARAMETERS
```

例 2 - REPLACE-TABLE アクションに対するパラメータ

```
PROCESS REPORT ACTION REPLACE-TABLE WITH
  PARAMETERS
    REPORT-ID = #HANDLE
    WORK-FILE = 5
  END-PARAMETERS
```

例 3 - SET-PRINTER アクションに対するパラメータ

```
PROCESS REPORTER ACTION SET-PRINTER WITH 'LPT1'
```

例 4 - SET-PRINT-OPTIONS アクションに対するパラメータ

```
DEFINE DATA LOCAL
  USING 'NGLUKEY1'
END-DEFINE
...
PROCESS REPORT ACTION SET-PRINT-OPTIONS WITH #HANDLE
  A4 0 0 0 0 0 0 FALSE FALSE FALSE FALSE FALSE AUTOMATIC
```

```
DEFINE DATA LOCAL
  USING 'NGLUKEY1'
END-DEFINE
...
PROCESS REPORT ACTION SET-PRINT-OPTIONS WITH PARAMETERS
  REPORT-ID = #HANDLE
  PAPER-SIZE = A4
  PAPER-WIDTH = 0
```

```
  PAPER-HEIGHT = 0
  LEFT-MARGIN = 0   TOP-MARGIN = 0
  RIGHT-MARGIN = 0  BOTTOM-MARGIN = 0
  LANDSCAPE = FALSE
  FAST-PRINT = FALSE
  SUPPRESS-BLANK-LINES = FALSE
  IGNORE-DUPLICATES = FALSE
  SHOW-PRINT-DIALOG = FALSE
  PAPER-SOURCE = AUTOMATIC
END-PARAMETERS
```

例 5 - CLOSE、PRINT、PREVIEW、EDIT アクションに対するパラメータ

```

PROCESS REPORT ACTION PRINT WITH #HANDLE
PROCESS REPORT ACTION PREVIEW WITH #HANDLE
PROCESS REPORT ACTION CLOSE WITH #HANDLE
PROCESS REPORT ACTION EDIT WITH #HANDLE
PROCESS REPORTER ACTION EDIT
```

102 PROPERTY

```
PROPERTY property-name
  OF [INTERFACE] interface-name
  IS operand
END-PROPERTY
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CREATE OBJECT** | **DEFINE CLASS** | **INTERFACE** | **METHOD** | **SEND METHOD**

関連機能グループ：「*コンポーネントベースプログラミング*」

機能

PROPERTY ステートメントでは、インターフェイス定義外で、実装としてオブジェクトデータ変数オペランドをプロパティに割り当てます。

問題のインターフェイス定義がコピーコードから取得され、クラス固有の方法で実装される場合は、これが使用されます。

DEFINE CLASS ステートメント内で、インターフェイス定義の後にのみ使用できます。

指定するインターフェイス名およびプロパティ名は、DEFINE CLASS ステートメントの INTERFACE 節で定義されている必要があります。

構文説明

property-name	これは、プロパティに割り当てられた名前です。
OF *interface-name*	これは、インターフェイスに割り当てられた名前です。
IS *operand*	IS節の *operand* により、オブジェクトデータ変数をプロパティ値の保存場所として割り当てます。
END-PROPERTY	PROPERTY ステートメントを終了するには、Natural 予約語 END-PROPERTY を使用する必要があります。

例

METHOD ステートメントのドキュメントに記載の例は、同じインターフェイスを 2 つのクラスに別々に実装する方法と、これを実現するために PROPERTY ステートメントと METHOD ステートメントを使用する方法を示しています。

103 READ

```
{ READ   }          [{ ALL          }]    [MULTI-FETCH-clause]
{ BROWSE }          [{ (operand1)   }]    [RECORDS] [IN] [FILE]
                                          view-name
                [PASSWORD=operand2]
                [CIPHER=operand3]
                [WITH REPOSITION]
                [sequence/range-specification]
                [STARTING WITH ISN=operand4]
                [WHERE  logical-condition]
                statement ...
END-READ        （ストラクチャードモードのみ）
[LOOP]          （レポーティングモードのみ）
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **GET TRANSACTION DATA** | **DELETE** | **END TRANSACTION** | **FIND** | **HISTOGRAM** | **GET** | **GET SAME** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **RETRY** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

READ ステートメントは、データベースからレコードを読み取るために使用します。 レコードは物理順、Adabas ISN 順、またはディスクリプタ（キー）フィールドの値順で取得できます。

このステートメントでは、処理ループを開始します。

「*READ ステートメント*」（『*プログラミングガイド*』）も参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S						N	P	I		B *						可	不可
operand2	C	S				A												可	不可
operand3	C	S						N										可	不可
operand4	C	S						N	P	I		B *						可	不可

* *operand1* および *operand4* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

operand1	読み取るレコード数： 読み取るレコード数は、*operand1* を（キーワード READ の直後にカッコで囲んで）指定することで制限できます。この際、キーワード READ の直後にカッコで囲んで数値定数（0～4294967295）または変数として指定します。 次に例を示します。 `READ (5) IN EMPLOYEES ...` `MOVE 10 TO CNT(N2)` `READ (CNT) EMPLOYEES ...` このステートメントでは、指定した制限が、LIMIT ステートメントで設定した制限より優先されます。 プロファイルまたはセッションパラメータ LT で設定された制限のほうが小さい場合、LT 制限が適用されます。 **注意:** 1. 読み取るレコード数が 4 桁の場合、先頭にゼロを付けて指定します（0*nnnn*）。カッコで囲まれた 4 桁の数はすべてステートメントの行番号参照として Natural で解釈されるためです。 2. *operand1* は、READ ループに入るときに評価されます。*operand1* の値が READ ループ内で変更されても、読み取られるレコード数には影響しません。
ALL	全レコードを読み取ることを強調するために、オプションでキーワード ALL を指定できます。

MULTI-FETCH-clause	以下の「*MULTI-FETCH 節*」を参照してください。
view-name	ビュー名： *view-name* として、ビューの名前を指定します。この名前は、DEFINE DATA ステートメント内、あるいはプログラム外のグローバルデータエリアまたはローカルデータエリアに事前に定義しておく必要があります。 レポーティングモードでは、*view-name* をDDMの名前にすることもできます。
PASSWORD **CIPHER**	**PASSWORD** および **CYPHER** 節： これらの節は Adabas データベースにのみ適用可能です。 Entire System Server では使用できません。 PASSWORD 節は、パスワード保護されているファイルからデータを取得するときのパスワードの指定に使用します。 CIPHER 節は、暗号化されているファイルからデータを取得するときのサイファキーの指定に使用します。 詳細については、ステートメント FIND および PASSW を参照してください。
WITH REPOSITION	このオプションは、READ ステートメントを再位置決めイベントに反応するように設定するために使用します。「*WITH REPOSITION オプション*」を参照してください。
sequence/range-specification	このオプションでは、取得順序／範囲を指定します。「*順序／範囲の指定*」を参照してください。
STARTING WITH ISN=*operand4*	この節は Adabas データベースにのみ適用されます。 **Adabas** へのアクセス この節は、物理順または論理順（昇順／降順）の READ ステートメントと組み合わせて使用します。 指定された値（*operand4*）は Adabas ISN（内部シーケンス番号）を表し、READ ループを開始するレコードを明確に指定するために使用します。 ■ 論理順 等号（=）で記述されても、READ ステートメントは対応するディスクリプタフィールドに同じ開始値を持つレコードのみを返すわけではなく、指定された開始値で始めて昇順または降順に論理的な参照を開始します。いくつかのレコードがディスクリプタフィールドに同じ内容を持っている場合、それらは ISN ソート順に返されます。 STARTING WITH ISN 節は、開始値が最初のレコードのディスクリプタ値と一致している場合にのみ適用される一種の「第 2 レベル」の選択条件になります。開始値と同じディスクリプタ値の全レコードおよび開始 ISN 「以下」（降順 READ については 「以

	上」）の ISN は Adabas で無視されます。 READ ループで返される最初のレコードは次のいずれかです。 ■ ディスクリプタ = 開始値の最初のレコードおよび開始 ISN より「高い」（降順 READ については 「低い」）ISN。 ■ このようなレコードが存在しない場合は、開始値より 「大きい」（降順 READ については 「小さい」）ディスクリプタを持つ最初のレコード。 ■ 物理順 レコードは、それらが物理的に保存されている順序で返されます。 STARTING WITH ISN 節を指定した場合、開始 ISN と同じ ISN のレコードに達するまで、Adabas では全レコードが無視されます。返される最初のレコードは、ISN = 開始 ISN のレコードに続く次のレコードです。 例 この節は、処理を継続する次のレコードを簡単に指定するために、処理が中断された READ ループ内での再位置決めに使用できます。これは特に、次のレコードをディスクリプタ値で一意に識別できない場合に役立ちます。 また、分散クライアント／サーバーアプリケーションで、レコードをサーバープログラムで読み取ってクライアントプログラムで処理し、レコードを順次処理ではなくバッチ処理する場合にも役立ちます。 例については、下記のプログラム **READISND** を参照してください。
WHERE *logical-condition*	「*WHERE 節*」を参照してください。
END-READ	READ ステートメントを終了するには、Natural 予約キーワード END-READ を使用する必要があります。

MULTI-FETCH 節

Note: この節は、Adabas データベースでのみ使用できます。

```
[ MULTI-FETCH { ON                          } ]
              { OFF                         }
              { OF multi-fetch-factor       }
```

Note: [MULTI-FETCH OF *multi-fetch-factor*] は、データベースタイプ ADA および ADA2 に対しては評価されません。 デフォルト処理モードが適用されます（プロファイルパラメータ MFSET を参照）。 MULTI-FETCH 節は、データベースタイプ ADA2 で使用しても完全に無視されます（『*コンフィグレーションユーティリティ*』ドキュメントの「*データベース管理システムの割り当て*」を参照）。

詳細については、『*プログラミングガイド*』の「*MULTI-FETCH 節*」（Adabas）を参照してください。

WITH REPOSITION オプション

Note: このオプションは、基準データベースが Adabas の場合にのみ適用されます。

WITH REPOSITION オプションでは、READ ステートメントを再位置決めイベントに反応するように設定できます。 これにより、アクティブな READ ループ内で別の開始値に再位置決めを行うことができます。 READ ステートメントの処理は新しい開始値で継続します。

READ ステートメントを WITH REPOSITION オプション付きで使用すると、再位置決めイベントが 2 つの方法のいずれかで起動されます。

1. ESCAPE TOP REPOSITION ステートメントが実行されるとき。 ESCAPE TOP REPOSITION ステートメントの実行時、Natural はループ開始への即時分岐を行い、再スタートを実行します。つまり、データベースは、検索値変数の現在の内容に従って、ファイルの新しいレコードに再位置決めを行います。 同時に、ループカウンタ *COUNTER をゼロにリセットします。
2. READ ループがデータベースから次のレコードを取り出そうとして、システム変数 *COUNTER の値が "0" であるとき。

   Note: アクティブな READ ループ内で *COUNTER が "0" に設定される場合、現在のレコードの処理は継続します。ループ開始への即時分岐は行われません。Natural for Windows、Natural for UNIX、および Natural for OpenVMS では、この方法で再位置決めイベントを起動することはできません。 この機能は、Natural for Mainframes バージョン 3.1 との互換性の理由のためにのみ保持されています。 したがって、このプロセスを使用することはお勧めしません。

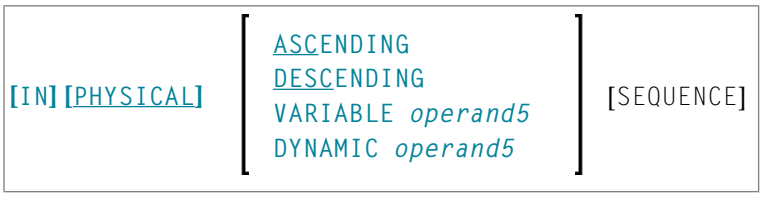
機能上の考慮事項

- READ ステートメントにループ制限（例：READ (10) EMPLOYEES WITH REPOSITION ..）があり、再スタートイベントが起動された場合、再位置決めが発生するまでにすでに処理されたレコード件数にかかわらず、ループは別の新しい 10 レコードを取得します。
- ESCAPE TOP REPOSITION ステートメントが実行される場合、ただし、（WITH REPOSITION キーワードが READ ステートメントに設定されていないか、事後ループステートメントが READ 以外であるために）最も内側のループが再位置決めできないときは、対応するランタイムエラーが発行されます。
- ESCAPE TOP ステートメントは参照を許可しないので、最も内側の処理ループが READ ..WITH REPOSITION ステートメントであれば、再位置決めイベントの開始だけができます。
- 再位置決めイベントは AT START OF DATA セクションの実行を起動せず、ループ制限オペランド（変数の場合）の再評価も起動しません。
- 検索値が変更されなかった場合、ループは最初のループ開始のように同じレコードに再位置決めします。

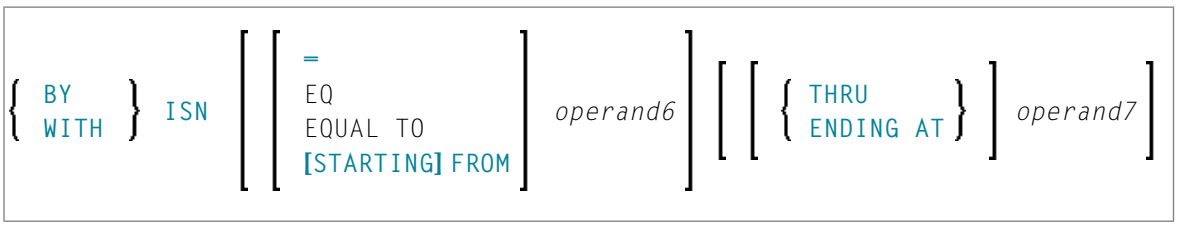
順序／範囲の指定

取得順序／範囲の指定には、3 つの構文オプションを使用できます。

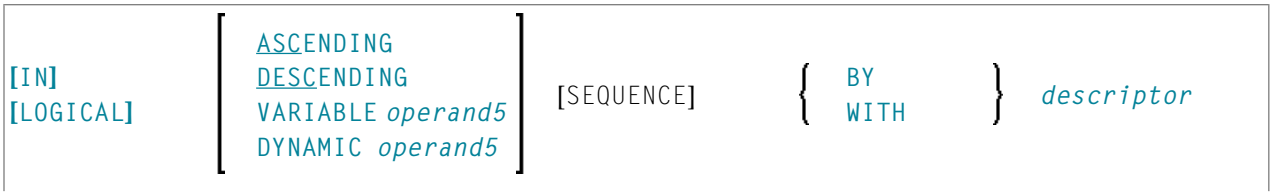
構文オプション 1：

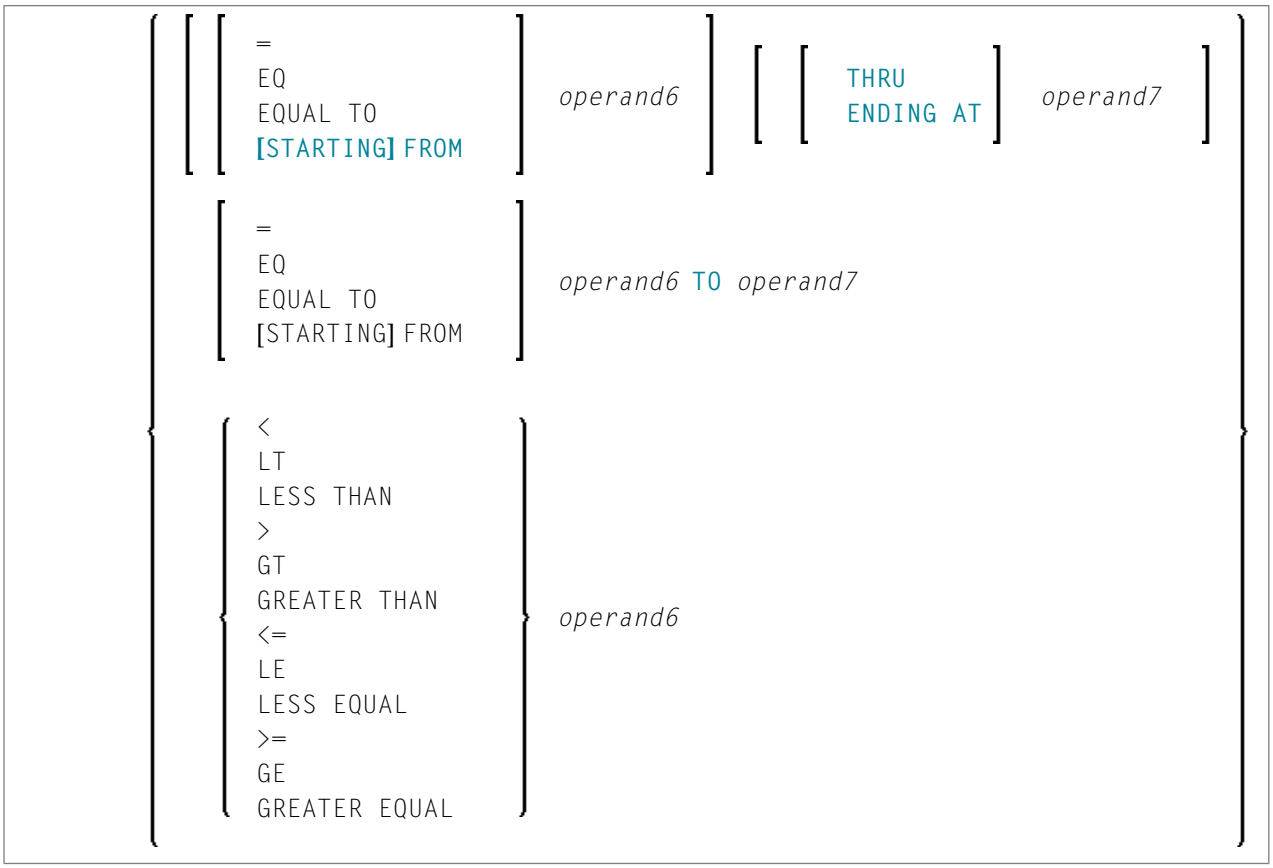


構文オプション 2：



構文オプション 3：





Notes:

1. Entire System Server で、構文オプション [2] と [3] は使用できません。
2. 図 3 の比較演算子を使用した場合は、オプション ENDING AT、THRU、および TO は使用できません。 これらの比較演算子は HISTOGRAM ステートメントにも有効です。

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand5		S				A												可	不可
operand6	C	S				A		N	P	I	F	B *	D	T	L			可	不可
operand7	C	S				A		N	P	I	F	B *	D	T	L			可	不可

* *operand6* および *operand7* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

READ IN PHYSICAL SEQUENCE	PHYSICAL SEQUENCE は、データベースに物理的に保存されている順にレコードを読み取る場合に使用します。 PHYSICAL はデフォルトの順序です。
READ BY ISN	READ BY ISN は、Adabas データベースに対してのみ使用できます。 READ BY ISN は、レコードを Adabas ISN（内部シーケンス番号）順に読み取るときに使用します。 **注意:** XML データベースの場合：READ BY ISN は Tamino オブジェクト ID の順に従って XML オブジェクトを読み取るために使用します。
READ IN LOGICAL SEQUENCE	LOGICAL SEQUENCE は、レコードをディスクリプタ（キー）フィールドの値の順に読み取るために使用します。 ディスクリプタが指定されると、レコードはそのディスクプタの値の順序で読み取られます。 順序制御には、ディスクリプタ、サブディスクリプタ、スーパーディスクリプタ、またはハイパーディスクリプタを使用できます。 フォネティックディスクリプタ、ピリオディックグループ内のディスクリプタ、またはピリオディックグループフィールドを含むスーパーディスクリプタは使用できません。 ディスクリプタを指定しなかった場合、DDM に指定されたデフォルトのディスクリプタ（"Default Sequence" フィールド）が使用されます。 順序制御に使用するディスクリプタが空値省略で定義されている場合（Adabas のみ）、ディスクリプタ値が空値であるレコードは読み取られません。 順序制御に使用するディスクリプタがマルチプルバリューフィールドの場合（Adabas のみ）、存在する値の個数に応じて同じレコードが何回も読み取られます。 **注意:** READ IN LOGICAL SEQUENCE については、『*プログラミングガイド*』でも説明されています。「*データベースアクセスのためのステートメント*」の「*READ ステートメント*」を参照してください。
ASCENDING | DESCENDING | VARIABLE | DYNAMIC SEQUENCE	この節は、Adabas、XML、および SQL データベースにのみ適用されます。READ PHYSICAL ステートメントでは、DB2 データベースにのみ適用できます。 この節を使用して、レコードを昇順または降順のどちらで読み取るかを指定できます。 ■ デフォルトは昇順です。昇順はキーワード ASCENDING を使用して明示的に指定することもできます。 ■ レコードを降順で読み取る場合は、キーワード DESCENDING を指定します。 ■ レコードを昇順または降順のどちらで読み取るかを事前に決めずに実行時に決めるには、変数（*operand5*）の前に VARIABLE キーワードまたは DYNAMIC キーワードを指定します。 *Operand5* は、フォーマット／長さ A1 で、値 "A"（「昇順」）または "D"（「降順」）が含まれている必要があります。

	■ キーワード VARIABLE を使用した場合、読み取り順（*operand5* の値）は、READ 処理ループの開始時に評価され、READ ループ内で *operand5* フィールドが変更されるかどうかに関係なく、ループが終了するまで変わりません。 ■ キーワード DYNAMIC を使用した場合、読み取り順（*operand5* の値）は、READ 処理ループでの各レコード取り出し前に評価され、レコードからレコードへ変更できます。これは、READ ループの任意の場所で昇順から降順（または、逆）へのスクロール順の変更を可能にします。 **注意:** 1. XML データベースの場合：DYNAMIC SEQUENCE は使用できません。
STARTING FROM ... ENDING AT/TO	STARTING FROM および ENDING AT 節は、ユーザー指定の値の範囲を基準にしてレコードの集合の読み取りを制限するために使用します。 STARTING FROM 節（=、EQ、EQUAL TO、または [STARTING] FROM）は、読み取り操作の開始値を決めます。 開始値を指定すると、指定した値から読み取りが始まります。 ファイルに開始値が存在しない場合、次に高い（または DESCENDING 読み取りでは低い）値が使用されます。 より高い（または DESCENDING ではより低い）値が存在しなければ、ループに入りません。 レコードを終了値に制限するには、包括的な範囲を示す用語 THRU、ENDING AT、または TO を持つ ENDING AT 節を指定できます。 読み取りディスクリプタフィールドが終了値を超えると、自動的なループ終了が実行されます。 TO、THRU および ENDING AT キーワードの基本機能は、非常によく似ていますが、内部的に動作が異なります。
THRU/ENDING AT	THRU または ENDING AT を使用した場合、開始値だけがデータベースに供給され、データベースによってレコードが返された後に、Natural ランタイムシステムによって終了値のチェックが実行されます。 読み取り順が ASCENDING の場合、開始値は READ ループで最初に返される値（およびレコード）を表すため、開始値としてより低い値を供給し、終了値としてより高い値を供給する必要があります。 ただし、逆順読み取り（DESCENDING）を呼び出す場合、より高い値は開始値に、そして終了値にはより低い値が出現する必要があります。 内部的に、読み取る範囲の終了を決定するために、Natural は終了値を超えるレコードを 1 件読み取ります。 終了値に到達したために READ ループを終了したとき、この最後のレコードは実際には要求した範囲内の最後のレコードではなく、その範囲を超えた最初のレコードである（最後の結果レコードを超えるレコードがファイルに含まれない場合を除く）ことに注意してください。 THRU および ENDING AT 節は、READ または HISTOGRAM ステートメントをサポートするすべてのデータベースに対して使用できます。
TO	キーワード TO を使用すると、開始値と終了値の両方がデータベースに送られ、Natural は値の範囲をチェックしません。 終了値を超えると、データベースは「ファイルの終わり」に到達した場合と同様に対処し、データベースループを終了します。 データベースにより完全な範囲チェックが行われるので、ASCENDING または DESCENDING のいずれの順序でブラウズしていても、常に範囲の小さい方の値が開始値になり、大きい方の値が終了値になります。

	TO オプションは、基準データベースが UNIX、OpenVMS、Windows 上の Adabas バージョン 3.1.1、メインフレーム上の Adabas バージョン 7（またはそれ以降）、Tamino、または SQL データベースの場合にのみ適用できます。

THRU/ENDING AT と TO の機能上の違いに関する注意

次のリストは、THRU/ENDING AT および TO オプションの機能上の違いを示しています。

THRU/ENDING AT	TO
終了値に達したために READ ループが終了するとき、ビューには「範囲外」の最初のレコードが含まれます。	終了値に達したために READ ループが終了するとき、ビューには指定した範囲の最後のレコードが含まれます。
終了値変数が READ ループ中に修正された場合、次のレコードの読み取り時に新しい値が終了値チェックに使用されます。	終了値変数は READ ループ開始時にのみ評価されます。 READ ループ中の以降の修正はすべて無効です。
範囲が不正な場合（例えば、READ .. = 'B' THRU 'A'）、レコードが返されないだけで、データベースエラーは発生しません。	値範囲を降順で供給してはならないため、不正な範囲はデータベースエラー（例：Adabas RC=61）となります。
READ .. DESCENDING が、開始値および終了値と一緒に使用される場合、開始値はファイルでの位置決めに使用され、終了値は Natural で「範囲終了」をチェックするために使用されます。したがって、開始値は終了値以上の値です。	両方の値がデータベースに渡されるため、それらは昇順で出現する必要があります。つまり、昇順または降順で読み取っているかどうかに関係なく、開始値は終了値以下の値です。
範囲のオーバーフローをチェックするために、ディスクリプタ値は基準データベースビューに出現する必要があります。つまり、それはレコードバッファに返される必要があります。	ディスクリプタは、返されるレコードフィールドに必要ではありません。
Adabas マルチバリューフィールド（MU フィールド）またはサブ／スーパー／ハイパーディスクリプタに対して終了値チェックはできません。また、プログラムコンパイル時に構文エラー NAT0160 を引き起こします。	MU フィールドおよびサブ／スーパー／ハイパーディスクリプタに対して終了値を指定できます。
すべてのデータベースに使用できます。	Windows、UNIX、OpenVMS 上の Adabas バージョン 3.1.1、メインフレーム上の Adabas バージョン 7（またはそれ以降）、Tamino、または SQL データベースに対してのみ使用できます。

WHERE 節

```
WHERE logical-condition
```

WHERE 節は、追加の選択条件（*logical-condition*）を指定するために使用できます。この条件は、値が読み取られた後、値に対する処理が実行される*前*に評価されます（AT BREAK 評価を含む）。

logical-condition の構文については、『*プログラミングガイド*』の「*論理条件基準*」で説明しています。

WHERE 節を含む READ ステートメントに LIMIT ステートメントや処理回数の制限を指定した場合、WHERE 節で排除されたレコードは制限数にカウントされません。

READ で使用可能なシステム変数

Natural システム変数 *ISN および *COUNTER が READ ステートメントで使用可能です。

これらのシステム変数は、P10 のフォーマットと長さで定義されています。 このフォーマット／長さは変更できません。

システム変数の使用方法について次に説明します。

***ISN**	システム変数 *ISN には、現在処理中のレコードの Adabas ISN が入ります。
***COUNTER**	システム変数 *COUNTER には、入力した処理ループの回数が含まれます。

例

- 例 1 - READ ステートメント
- 例 2 - READ WITH REPOSITION
- 例 3 - READ および FIND ステートメントの結合
- 例 4 - DESCENDING オプション
- 例 5 - VARIABLE オプション
- 例 6 - DYNAMIC オプション

例 1 - READ ステートメント

```
** Example 'READEX1S': READ (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 3
*
WRITE 'READ IN PHYSICAL SEQUENCE'
READ EMPLOY-VIEW IN PHYSICAL SEQUENCE
  DISPLAY NOTITLE PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
WRITE / 'READ IN ISN SEQUENCE'
READ EMPLOY-VIEW BY ISN STARTING FROM 1 ENDING AT 3
  DISPLAY         PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
WRITE / 'READ IN NAME SEQUENCE'
READ EMPLOY-VIEW BY NAME
  DISPLAY         PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
WRITE / 'READ IN NAME SEQUENCE STARTING FROM ''M'''
READ EMPLOY-VIEW BY NAME STARTING FROM 'M'
  DISPLAY         PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
END
```

プログラム **REAEX1S** の出力：

```
PERSONNEL         NAME            ISN         CNT
   ID
--------- -------------------- ----------- -----------

READ IN PHYSICAL SEQUENCE
50005800  ADAM                           1           1
50005600  MORENO                         2           2
```

```
50005500  BLOND                          3           3

READ IN ISN SEQUENCE
50005800  ADAM                           1           1
50005600  MORENO                         2           2
50005500  BLOND                          3           3

READ IN NAME SEQUENCE
60008339  ABELLAN                      478           1
30000231  ACHIESON                     878           2
50005800  ADAM                           1           3

READ IN NAME SEQUENCE STARTING FROM 'M'
30008125  MACDONALD                    923           1
20028700  MACKARNESS                   765           2
40000045  MADSEN                       508           3
```

レポーティングモードの例はライブラリ SYSEXRM のプログラム **REAEX1R** を参照してください。

例 2 - READ WITH REPOSITION

```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR     (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
INPUT (IP=OFF AD=O) 'NAME:' NAME /
    'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
    'Press PF3 to stop'
  IF *PF-KEY = 'PF3'
    THEN STOP
  END-IF
  IF #ATTR MODIFIED
    THEN ESCAPE TOP REPOSITION
  END-IF
```

```
END-READ
...
```

```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR     (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
  INPUT (IP=OFF AD=O) 'NAME:' NAME /
    'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
    'Press PF3 to stop'
  IF *PF-KEY = 'PF3'
    THEN STOP
  END-IF
  IF #ATTR MODIFIED
    THEN RESET *COUNTER
  END-IF
END-READ
...
```

例 3 - READ および FIND ステートメントの結合

次の例では、まずEMPLOYEESファイルから、ディスクリプタNAMEの値で論理順にレコードを読み取ります。 次に、EMPLOYEES ファイルの PERSONNEL-ID（人事番号）を検索条件として、VEHICLESファイルをFINDステートメントで検索します。 結果のレポートには、読み取られた従業員の名前（EMPLOYEES ファイルから読み取られる）と、この従業員が所有する自動車のモデル（VEHICLES ファイルから読み取られる）が示されます。 1 人で何台も自動車を所有している場合、同一名で複数行が出力されます。

```
** Example 'REAEX2': READ and FIND combination
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 10
```

```
*
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  SUSPEND IDENTICAL SUPPRESS
  FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      ENTER
    END-NOREC
    DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
            PERSONNEL-ID (RD.)
            FIRST-NAME (RD.)
            MAKE (FD.) (IS=OFF)

  END-FIND
END-READ
END
```

プログラム **REAEX2** の出力：

```
   PERSONNEL      FIRST-NAME         MAKE
      ID
--------------- --------------- ---------------

20007500        VIRGINIA        CHRYSLER
20008400        MARSHA          CHRYSLER
                                CHRYSLER
20021100        ROBERT          GENERAL MOTORS
20000800        LILLY           FORD
                                MG
20001100        EDWARD          GENERAL MOTORS
20002000        MARTHA          GENERAL MOTORS
20003400        LAUREL          GENERAL MOTORS
30034045        KEVIN           DATSUN
30034233        GREGORY         FORD
11400319        MANFRED
```

例 4 - DESCENDING オプション

```
** Example 'READSCND': READ (with DESCENDING SEQUENCE)
************************************************************************
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
END-DEFINE
*
READ (10) EMPL IN DESCENDING SEQUENCE BY NAME FROM 'ZZZ'
  DISPLAY *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD)
```

```
END-READ
END
```

例 5 - VARIABLE オプション

```
** Example 'REAVSEQ':  READ  (with VARIABLE SEQUENCE)
************************************************************************
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #DIR        (A1)
1 #STARTVALUE (A20)
END-DEFINE
*
SET KEY PF7 PF8
*
INPUT 'Select READ direction'
   // 'Press' 08T 'PF7' (I)                  21T 'to read backward'
    /          08T 'PF8' (I) 'or' 'ENTER' (I) 21T 'to read forward'
*
IF *PF-KEY = 'PF7'
  MOVE 'D'   TO #DIR
  MOVE 'ZZZ' TO #STARTVALUE
ELSE
  MOVE 'A' TO #DIR
  MOVE 'A' TO #STARTVALUE
END-IF
*
READ (10) EMPL IN VARIABLE #DIR SEQUENCE
               BY NAME FROM #STARTVALUE
  DISPLAY *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD)
END-READ
END
```

例 6 - DYNAMIC オプション

```
DEFINE DATA LOCAL
1 #DIRECTION (A1) INIT <'A'>   /* 'A' = ASCENDING
1 #EMPVIEW VIEW OF EMPLOYEES
2 NAME
...
END-DEFINE
...
READ #EMPVIEW IN DYNAMIC #DIRECTION SEQUENCE BY NAME = 'SMITH'
   INPUT (AD=O) NAME
```

```
       / 'Press PF7 to scroll in DESCENDING sequence'
       / 'Press PF8 to scroll in ASCENDING  sequence'
   ..
   IF *PF-KEY = 'PF7' THEN MOVE 'D' TO #DIRECTION END-IF
   IF *PF-KEY = 'PF8' THEN MOVE 'A' TO #DIRECTION END-IF
END-READ
...
```

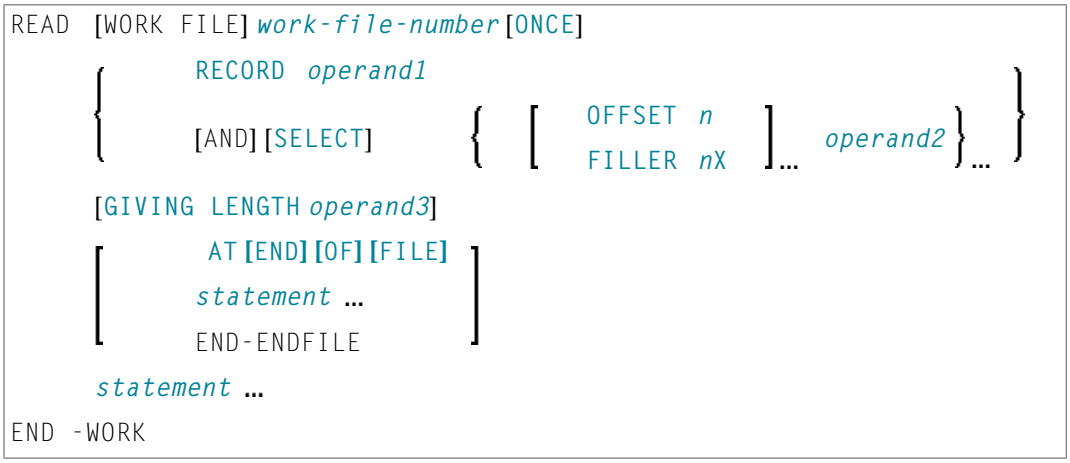
例 7 - STARTING WITH ISN 節

```
** Example 'READISND': READ  (with STARTING WITH ISN)
************************************************************************
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #DIR      (A1)
1 #STARTVAL (A20)
1 #STARTISN (N8)
END-DEFINE
*
SET KEY PF3 PF7 PF8
*
MOVE 'ADKINSON' TO #STARTVAL
*
READ (9) EMPL BY NAME = #STARTVAL
  WRITE *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD) *COUNTER
  IF *COUNTER = 5 THEN
    MOVE NAME TO #STARTVAL
    MOVE *ISN TO #STARTISN
  END-IF
END-READ
*
#DIR := 'A'
*
REPEAT
 READ EMPL IN VARIABLE #DIR  BY NAME = #STARTVAL
            STARTING WITH ISN = #STARTISN
    MOVE NAME TO #STARTVAL
    MOVE *ISN TO #STARTISN
    INPUT NO ERASE  (IP=OFF AD=O)
         15/01 *ISN  NAME  FIRST-NAME  BIRTH (EM=YYYY-MM-DD)
           //  'Direction:' #DIR
           //  'Press PF3 to stop'
           /   '      PF7 to go step back'
           /   '      PF8 to go step forward'
           /   '      ENTER to continue in that direction'
    /*
```

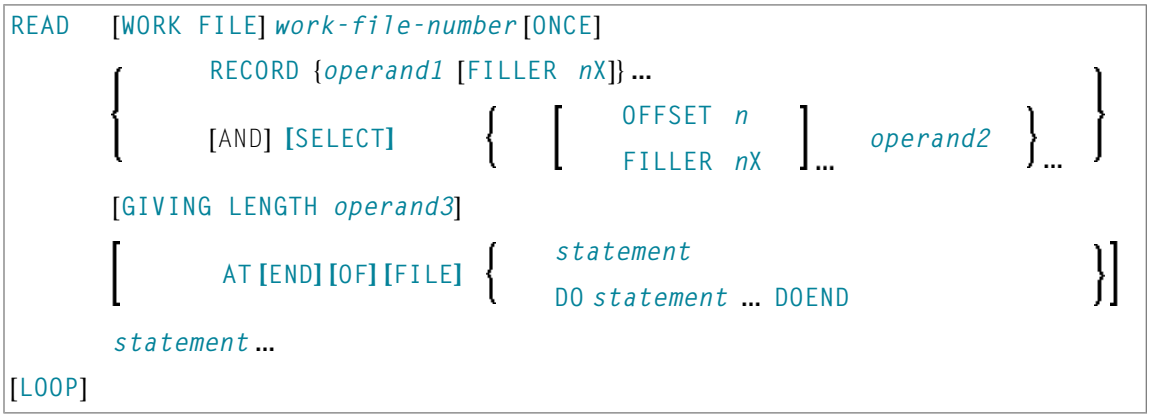
```
    IF *PF-KEY = 'PF7' AND #DIR = 'A'
      MOVE 'D' TO #DIR
      ESCAPE BOTTOM
    END-IF
    IF *PF-KEY = 'PF8' AND #DIR = 'D'
      MOVE 'A' TO #DIR
      ESCAPE BOTTOM
    END-IF
    IF *PF-KEY = 'PF3'
      STOP
    END-IF
  END-READ
  /*
  IF *COUNTER(0290) = 0
    STOP
  END-IF
END-REPEAT
END
```

104 READ WORK FILE

ストラクチャードモード構文



レポーティングモード構文



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CLOSE WORK FILE** | **DEFINE WORK FILE** | **WRITE WORK FILE**

関連機能グループ：「*ワークファイル／PC ファイルの制御*」

機能

READ WORK FILE ステートメントは、Adabas 以外の物理順ワークファイルからデータを読み取るために使用します。 データはワークファイルから順次読み取られます。 データがどのように読み取られるかは、そのデータがワークファイルにどのように書き込まれたということとは無関係です。

READ WORK FILE は、ワークファイルの全レコードを読み取るための処理ループを開始し、実行します。 READ WORK FILE ループでは自動ブレイク処理を行うことができます。

Notes:

1. READ WORK FILE ステートメントの実行中にエンドオブファイル条件になると、Natural はワークファイルを自動的にクローズします。
2. Entire Connection の場合：Entire Connection ワークファイルを読み取る場合、READ WORK FILE 処理ループ内に I/O ステートメントを指定しないでください。
3. Unicode およびコードページのサポートについては、『*Unicode およびコードページのサポート*』ドキュメントの「*Windows、UNIX、および OpenVMS プラットフォーム上のワークファイルと出力ファイル*」を参照してください。

ASCII ワークファイルを読み取る場合、最後の物理レコードの後に最後のレコードとして空のレコードが返されることがあります。 Natural は個々のレコードを読み取るわけではなく、ファイルアクセスパフォーマンスを最適化するためにワークファイルのより大きなブロックを読み取るためです。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1		S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	可	可
operand2		S	A	G		A	U	N	P	I	F	B	D	T	L	C		可	可
operand3		S								I								可	可

ワークファイルタイプ ENTIRECONNECTION または TRANSFER を使用しているとき、*operand2* のフォーマットを C にすることはできません。

「*フィールド長*」も参照してください。

構文要素の説明：

work-file-number	ワークファイル番号： （Natural に定義した）読み取るワークファイルの番号。 変数インデックス範囲： ワークファイルから配列を読み取る場合、配列に変数インデックス範囲を指定できます。 次に例を示します。 `READ WORK FILE work-file-number #ARRAY (I:J)`
ONCE	**ONCE** オプション： ONCE は、1 レコードのみを読み取るために使用します。 処理ループは開始されません（したがってループを閉じるキーワード END-WORK または LOOP は指定できません）。 ONCE を指定する場合、AT END OF FILE 節も使用する必要があります。 ONCE 指定のある READ WORK FILE ステートメントがユーザー開始の処理ループで制御される場合、処理ループの終了前にワークファイルにエンドオブファイル条件が検出されることもあります。ワークファイルから読み取られたすべてのフィールドには、最後に読み取られたレコードの値が残っています。その後、ワークファイルは、READ WORK FILE ONCE の次回実行時に読み取られる最初のレコードに再位置決めされます。
RECORD *operand1* **FILLER** *n***X**	**RECORD** オプション： RECORD を指定した場合、読み取られた各レコードの全フィールドが以降の処理で使用可能になります。 レコードのレイアウトに対応するオペランドリスト（*operand1*）を提供する必要があります。 FILLER *n*X の指定があると、入力データ内で *n* バイト分がスキップされます。RECORD 節で定義されたレコードは、連続ストレージ内にある必要があります。FILLER はストラクチャードモードでは使用できません。 ストラクチャードモードまたは DEFINE DATA ステートメントで定義したレコードを使用する場合、1つのフィールド（またはグループ）だけが使用できます。 この場合、FILLER は指定できません。 レコードに含まれているデータが Natural によってチェックおよび変換されることはありません。数値フィールド内に数値以外のデータが含まれていると、プログラムの異常終了を起こすので、ユーザーは正しくレコードレイアウトを記述する必要があります。 Natural によるチェックは行われないため、このオプションはシーケンシャルファイルのレコードを処理するには最も速い方法です。レコードが読み取られる前は、*operand1* で定義したレコードエリアは空白で埋められます。 したがって、エンドオブファイル条件では空白のエリアが返されます。短いレコードには空白が付加されます。 下記の「*RECORD オプションの使用方法の概要*」を参照してください。
SELECT	**SELECT** オプション（デフォルト）：

	SELECT を指定した場合、オペランドリスト（ *operand2*）に指定されたフィールドのみが使用可能になります。 入力レコード内のフィールドの位置を OFFSET または FILLER 指定で示すことができます。	
	OFFSET *n*	OFFSET 0 は、レコードの先頭バイトを示します。
	FILLER *n*X	入力レコード内で *n* バイト分スキップすることを示します。
	Natural は、各フィールドに選択された値を割り当て、レコードから選択された数値フィールドに定義どおり正しい数値データが含まれているかをチェックします。 選択されたフィールドのチェックは Natural で行われるため、このオプションを使用すると、シーケンシャルファイルの処理にオーバーヘッドがかかります。 レコードが SELECT オプションで指定した全フィールドを満たさない場合、次のことが適用されます。 ■ フィールドの一部だけが満たされる場合、残りの部分は空白またはゼロにリセットされます。 ■ まったく満たされなかったフィールドは前の内容のままです。 ファイルタイプ CSV を読み取る場合、OFFSET オプションは無視されます。	
GIVING LENGTH *operand3*	GIVING LENGTH 節は、読み取るレコードの実際の長さを取得するために使用します。 長さ（バイト数）は*operand3* に返されます。 *operand3* は、フォーマット／長さ I4 で定義する必要があります。 ワークファイルが TYPE UNFORMATTED として定義されている場合、返される長さはバイトストリームから読み取られるバイト数（FILLER オペランドでスキップしたバイトを含む）を示します。 ワークファイルタイプ CSV で GIVING LENGTH 節を使用した場合は、GIVING LENGTH で指定したオペランドではレコード内のフィールド数（レコードの長さではない）が返されます。	
AT END OF FILE	AT END OF FILE 節は ONCE オプションとともに使用できます。 ONCE オプションを使用する場合、エンドオブファイル条件になったときに行う処理をこの節で指定する必要があります。 ONCE オプションを使用しない場合、エンドオブファイル条件は通常の処理ループ終了と同様に扱われます。	
END -WORK	READ WORK FILE ステートメントを終了するには、Natural 予約語 END-WORK を使用する必要があります。	

RECORD オプションの使用方法の概要

RECORD オプションとの併用	コンパイル時に排除	ランタイムに排除	RECORDオプションが無視され、処理がSELECTモードに切り替わる
ワークファイルタイプ ENTIRE CONNECTION		x	
ダイナミック変数	x		
ワークファイルタイプ CSV			x
ワークファイルタイプ PORTABLE			x
ワークファイルタイプ ASCII、ASCII COMPRESSED、CSV、UNFORMATTED、コンフィグレーションユーティリティでコードページが指定されている場合（変換が必要）、または1つ以上のUnicodeフィールドが指定されている場合（フォーマット U のオペランド、変換が必要）			x

フィールド長

オペランド定義テーブル内のフィールド長は次のように決定されます。

フォーマット	長さ
A、B、I、F	入力レコード中のバイト数は、内部長さ定義と同じです。
N	入力レコード中のバイト数は、小数点の前後の内部桁数の合計です。小数点と符号は、入力レコード中のバイト数には入りません。
P、D、T	入力レコード中のバイト数は、小数点の前後の桁数の合計に 1（符号用）を加算し、それを 2 で割って切り上げたものです。
L	1 バイトが使用されます。C フォーマットのフィールドについては、2 バイトが使用されます。

フィールド長の例：

フィールド定義	入力レコード
#FIELD1 (A10)	10 バイト
#FIELD2 (B15)	15 バイト
#FIELD3 (N1.3)	4 バイト
#FIELD4 (N0.7)	7 バイト
#FIELD5 (P1.2)	2 バイト
#FIELD6 (P6.0)	4 バイト

『*プログラミングガイド*』の「*ユーザー定義変数のフォーマットおよび長さ*」も参照してください。

ラージおよびダイナミック変数の処理

ワークファイルタイプ	処理
ASCII **ASCII-COMPRESSED** **SAG**（バイナリ）	ワークファイルタイプ ASCII、ASCII-COMPRESSED、および SAG（バイナリ）では、ダイナミック変数は処理できず、エラーが生成されます。ただし、最大フィールド／レコード長が 32766 バイトのラージ変数は処理できます。
ENTIRECONNECTION	ワークファイルタイプ ENTIRECONNECTION では、ダイナミック変数は処理できません。 ただし、このタイプでは、最大フィールド／レコード長が 107341824 バイトのラージ変数を処理できます。 どのダイナミック変数を使用する場合でも、RECORD オプションは許可されません。
PORTABLE **UNFORMATTED**	2 つのワークファイルタイプ PORTABLE と UNFORMATTED を使用して、ラージおよびダイナミック変数をワークファイルに書き込んだりワークファイルから読み取ったりできます。これらのタイプには、ダイナミック変数に対するサイズ制限がありません。ただし、ラージ変数は最大フィールド／レコード長の 32766 バイトを超えることはできません。 PORTABLE ワークファイルからダイナミック変数を読み取ると、保存されている長さへのサイズ変更が行われます。
CSV	ダイナミックおよびラージ変数の最大フィールド／レコード長は 32766 バイトです。 ダイナミック変数がサポートされます。 X-array は許可されていないので、エラーメッセージが表示されます。

例

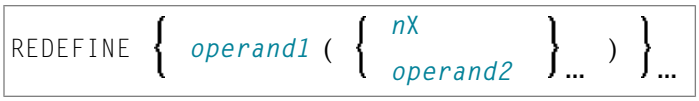
```
** Example 'RWFEX1': READ WORK FILE
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
*
1 #RECORD
  2 #PERS-ID (A8)
  2 #NAME    (A20)
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
  WRITE WORK FILE 1
        PERSONNEL-ID NAME
END-FIND
*
```

```
* ...
*
READ WORK FILE 1 RECORD #RECORD
  DISPLAY NOTITLE #PERS-ID #NAME
END-WORK
*
END
```

プログラム **RWFEX1** の出力：

```
#PERS-ID        #NAME
-------- --------------------

11100328 BERGHAUS
11100329 BARTHEL
11300313 AECKERLE
11300316 KANTE
11500304 KLUGE
11500308 DIETRICH
11500318 GASSNER
11500343 ROEHM
11600303 BERGER
11600320 BLAETTEL
11500336 JASPER
11100330 BUSH
11500328 EGGERT
```

105 REDEFINE



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

REDEFINE ステートメントは、フィールドを再定義するために使用します。 再定義により、1 つ以上のユーザー定義変数にできます。

1 つの REDEFINE ステートメントで複数のフィールドを再定義できます。

制限事項

REDEFINE ステートメントはレポーティングモードでのみ有効です。 ストラクチャードモードでフィールドを再定義するには、DEFINE DATA ステートメントの REDEFINE 節を使用します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1		S	A	G		A	U	N	P	I	F	B	D	T	L	C			可	不可
operand2		S	A	G		A		N	P	I	F	B	D	T	L	C			可	可

構文要素の説明：

REDEFINE	再定義の方法
operand1	*operand1* のバイト位置をフォーマットに関係なく左から右に再定義します。
operand2	*operand2* のフォーマットには、*operand1* と異なるフォーマットを指定できます。 REDEFINE ステートメントで指定するバイトの位置は、元のフィールド内のデータと合っている必要があります。 英数字フィールドが数字フィールドに再定義されて、フォーマット指定に従った数字データが入っていない場合、そのフィールドが使用されるときに異常終了することがあります。

	再定義したフィールドの再定義 REDEFINE ステートメントで定義したフィールドを、続いて別の REDEFINE ステートメントで再定義することもできます。
***n*X**	フィラー指定 *n*X 指定は、再定義されるフィールド／変数内のフィラーバイトを示します。 フィールド末尾の *n*X 指定は任意です。

例

例 1

ユーザー定義変数 #A（フォーマット／長さ A10）の値は "123ABCDEFG" です。

```
REDEFINE #A (#A1(N3) #A2(A7))
```

#A1 の値は "123" です。 #A2 の値は "ABCDEFG" です。

例 2

ユーザー定義変数#B（フォーマット／長さ A10）の値（16 進形式）は "12345CC1C2C3C4C5C6C7" です。

```
REDEFINE #B (#B1(P4) #B2(A7))
```

#B1 の値は "12345C"（16 進形式）です。

#B2 の値は "C1C2C3C4C5C6C7"（16 進形式）です。

```
REDEFINE #B (#BB1(B2)8X)) or REDEFINE #B(#BB1(B2))
```

#BB1 の値は "1234"（16 進形式）です。

Note: パックデータ（P フォーマット）では必要な桁数を指定します。次の式でパック形式のデータのバイト数を求めることができます。

バイト数 = （桁数 + 1） / 2、小数点以下は切り上げ

例 3

```
COMPUTE #V (N8.2) = #Y (N10) = ...
REDEFINE #V (3X  #A(N3)  2X  #P (N2))  #Y (#B(N3)  7X)
```

例 4

この例では、"YYYYMMDD" 形式のシステム変数 *DATN の値を再定義し、「日／月／年」 の順で 3 フィールドに分割して結果を出力しています。

```
MOVE *DATN TO #DATINT (N8)
REDEFINE #DATINT (#YEAR (N4) #MONTH (N2) #DAY (N2))
DISPLAY NOTITLE #DATINT #DAY #MONTH #YEAR
END
```

出力：

```
 #DATINT  #DAY #MONTH #YEAR
--------- ---- ------ -----

 19950108   8    1     1995
```

106 REDUCE

```
REDUCE { dynamic-clause } [GIVING operand5]
       { array-clause   }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**EXPAND** | **RESIZE**

関連機能グループ：「*ダイナミック変数または X-array のメモリ管理制御*」

機能

REDUCE ステートメントは、次を縮小するために使用します。

- ダイナミック変数（*dynamic-clause*）の割り当てられた長さ、または
- X-array（*array-clause*）のオカレンス数

詳細については、『*プログラミングガイド*』の「*ダイナミック変数の使用*」、「*X-array*」、「*X-Group 配列のストレージ管理*」も参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1		S	A			A	U					B							不可	不可
operand2	C	S								I									不可	不可
operand3			A	G		A		N	P	I	F	B	D	T	L	C	G	O	可	不可
operand4	C	S					U	N	P	I									不可	不可
operand5		S								I4									不可	可

構文要素の説明：

dynamic-clause	REDUCE DYNAMIC VARIABLE ステートメントは、ダイナミック変数（*operand1*）の割り当てられた長さを、指定した長さ（*operand2*）に縮小します。 詳細については、後述の「***Dynamic** 節*」を参照してください。
operand1	*operand1* は、長さが縮小されるダイナミック変数です。
operand2	*operand2* は、縮小後のダイナミック変数の長さを指定するために使用します。 値として、負ではない整数の定数または Integer4（I4）タイプの変数を指定する必要があります。
array-clause	REDUCE ARRAY ステートメントは、X-array（*operand3*）のオカレンス数を、*(dim[,dim[,dim]])* で指定した上下限に縮小します。 詳細については、以下の「***Array** 節*」を参照してください。
operand3	*operand3* は X-array です。 X-array のオカレンスを削減できます。 配列のインデックス表記はオプションです。 各次元でインデックス表記として使用できるのは、全範囲を示す表記 "*" のみです。
dim *operand4*	X-array 縮小の上下限表記（*operand4* またはアスタリスク）は、ここで指定します。 現在の上下限の値を使用する必要がある場合は、*operand4* の代わりにアスタリスク（*）を指定する必要があります。 詳細については、後述の「*次元*」を参照してください。
GIVING *operand5*	GIVING 節を指定しない場合は、エラー発生時に Natural ランタイムエラー処理がトリガされます。 GIVING 節を指定した場合は、*operand5* に、エラー発生時は Natural メッセージ番号、成功時はゼロが含まれます。

Dynamic 節

```
[SIZE OF] DYNAMIC [VARIABLE] operand 1 TO operand2
```

REDUCE DYNAMIC VARIABLE ステートメントは、ダイナミック変数（*operand1*）の割り当てられた長さを、指定した長さ（*operand2*）に縮小します。 指定した長さを超える分のダイナミック変数の割り当て済みメモリは、ステートメントの実行時に即座に解放されます。

ダイナミック変数の現在割り当てられている長さ（*LENGTH）が指定した長さより長い場合、*LENGTH は指定した長さに設定され、変数の内容が切り捨てられます（修正は行われません）。 指定した長さがダイナミック変数の現在割り当てられている長さより長い場合、ステートメントは無視されます。

Array 節

```
[OCCURRENCES OF] ARRAY operand3 TO { 0 | (dim[,dim[,dim]]) }
```

REDUCE ARRAY ステートメントは、X-array（*operand3*）のオカレンス数を、TO (*dim*[,*dim*[,*dim*]]) で指定した上下限に縮小します。

REDUCE TO 0（ゼロ）を指定した場合、X-array の全オカレンスが解放されます。 つまり、配列全体が削減されます。

REDUCE ステートメントで使用される上限または下限は、配列に定義された対応する上限または下限と正確に同じである必要があります。

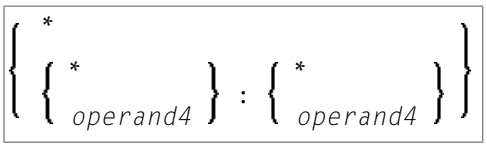
例：

```
DEFINE DATA LOCAL
1 #a(I4/1:*)
1 #g(1:*)
  2 #ga(I4/1:*)

1 #i(i4)
END-DEFINE
...


*/ reducing #a (1:10)

REDUCE ARRAY #a TO (1:10)       /* #a is reduced
REDUCE ARRAY #a TO (*:10)       /* to 10 occurrences.


*/ reducing #ga (1:10,1:20)

REDUCE ARRAY #g TO (1:10)       /* 1st dimension is set to (1:10)
REDUCE ARRAY #ga TO (*:*,1:20)  /* 1st dimension is dependent and
                                /* therefore kept with (*:*)
                                /* 2nd dimension is set to (1:20)

REDUCE ARRAY #a TO (5:10)       /* This is rejected because the lower index
                                /* must be 1 or *
REDUCE ARRAY #a TO (#i:10)      /* This is rejected because the lower index
                                /* must be 1 or *

REDUCE ARRAY #ga TO (1:10,1:20) /* (1:10) for the 1st dimension is rejected
                                /* because the dimension is dependent and
                                /* must be specified with (*:*).
```

詳細については、次を参照してください。

- *X-array のストレージ管理*
- *X-Group 配列のストレージ管理*

次元



X-array 縮小の上下限表記（*operand4* またはアスタリスク）は、ここで指定します。 現在の上下限の値を使用する必要がある場合は、*operand4* の代わりにアスタリスク（*）を指定できます。 "*:*" の代わりに単一のアスタリスクを指定できます。

次元数（*dim*）は、X-array（1、2、または 3）と正確に一致している必要があります。

REDUCE ステートメントを使用するときは、オカレンス数を減らすことだけが可能です。 要求した数が現在割り当てられているオカレンス数よりも大きい場合は無視されます。

107 REINPUT

```
REINPUT   [FULL] [(statement-parameters)] { USING HELP
                                          { WITH-TEXT-option }
          [MARK-option]
          [ALARM-option]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DEFINE WINDOW** | **INPUT** | **SET WINDOW**

関連機能グループ：「*対話型処理用の画面生成*」

機能

REINPUTステートメントは、INPUTステートメントに戻って再実行するために使用します。これは通常、前のINPUTステートメントの結果、データ入力が無効であったことを示すメッセージを表示するために使用します。「*例 1*」を参照してください。

INPUTステートメントとそれに対応するREINPUTステートメントの間に、WRITEまたはDISPLAYステートメントを実行することはできません。REINPUTステートメントをバッチモードで使用することはできません。

REINPUTステートメントが実行されると、INPUTステートメントが実行されたときに存在していたサブルーチン、特殊条件、およびループ処理を考慮してプログラムステータスが再位置づけされます（INPUTステートメントのステータスがまだ有効な場合）。INPUTステートメントの実行後にループが開始され、REINPUTステートメントがそのループ内にある場合、ループは中断され、REINPUTの結果として再処理されたINPUTステートメントの後で再開されます。

INPUTステートメントの実行後にサブルーチン階層が呼び出され、サブルーチン内でREINPUTが実行された場合、Naturalは自動的に全サブルーチンをさかのぼり、INPUTステートメントの位置にプログラムステータスを再位置づけします。

INPUTステートメントが実行されたステータスがすでに終了してしまった時点では、INPUTステートメントをループ、サブルーチン、または特殊条件ブロック内に位置づけ、REINPUTステートメントを実行することはできません。このエラー条件が検出されると、エラーメッセージが生成され、プログラムの実行が終了します。

Note: FULLオプションなしでREINPUTステートメントを実行しても、対応するINPUTステートメントで使用された属性制御変数の "MODIFIED" ステータスはリセットされません。属性制御変数にステータス "MODIFIED" が割り当てられているかどうかを確認するには、MODIFIEDオプションを使用します。

構文説明

<table>
<tr><td>REINPUT FULL</td><td>REINPUT ステートメントで FULL オプションを使用すると、対応する INPUT ステートメントが完全に再実行されます。

■ FULL オプションを使用しない通常の REINPUT ステートメントでは、INPUT ステートメントと REINPUT ステートメントの間で変更された変数の内容は表示されません。つまり、画面上のすべての変数は、INPUT ステートメントが初めて実行されたときに保持していた内容を示します。
■ REINPUT FULL ステートメントでは、INPUT ステートメントの最初の実行後に行われたすべての変更が、INPUT ステートメントの再実行時に INPUT ステートメント適用されます。つまり、画面上のすべての変数は、REINPUT ステートメントの実行時に保持していた値を含みます。

注意: REINPUT FULL によって、入力専用フィールド（AD=A）の内容は再度削除されます。

REINPUT FULL ステートメントの別の特徴は、属性制御変数のステータスが "NOT MODIFIED" にリセットされることです。 この処理は普通の REINPUT ステートメントでは行われません。 属性制御変数にステータス "MODIFIED" が割り当てられているかどうかを確認するには、MODIFIED オプションを使用します。

「例 3 - MARK POSITION を指定した REINPUT FULL」も参照してください。</td></tr>
<tr><td>statement-parameters</td><td>REINPUT ステートメントで指定されるパラメータは、ステートメントに指定した全フィールドに適用されます。

フィールドレベルで指定したパラメータ（MARK オプションを参照）は、ステートメントレベルの対応する各パラメータを上書きします。
<table>
<tr><th colspan="2">REINPUT ステートメントで指定可能なパラメータ</th><th>指定（S = ステートメントレベル、E = 要素レベル）</th></tr>
<tr><td>AD</td><td>属性定義 *</td><td>SE</td></tr>
<tr><td>CD</td><td>カラー定義</td><td>S</td></tr>
</table>
*AD=P をステートメントパラメータとして指定した場合、すべてのフィールド（MARK オプションで使用されるフィールドを除く）が保護されます。

各セッションパラメータの詳細については、『パラメータリファレンス』を参照してください。</td></tr>
<tr><td>USING HELP</td><td>USING HELP 節により、INPUT マップに定義されているヘルプルーチンが呼び出されます。

USING HELP を MARK オプション付きで使用すると、MARK オプションに指定された最初のフィールドに対して定義されているヘルプルーチンが呼び出され</td></tr>
</table>

	ます。 フィールドにヘルプルーチンが定義されていない場合は、マップ用のヘルプルーチンが呼び出されます。 例： `REINPUT USING HELP MARK 3` 結果として、INPUT マップの 3 番目のフィールドに対して定義されているヘルプルーチンが呼び出されます。
WITH-TEXT-option	WITH TEXT オプションは、メッセージ行に表示されるテキストを提供するために使用します。 下記の「***WITH TEXT オプション***」を参照してください。
MARK-option	MARKオプションを使用して、特定のフィールドをマークすることができます。つまり、REINPUT ステートメントの実行時にカーソルが配置されるフィールドを指定できます。 下記の「***MARK オプション***」を参照してください。
ALARM-option	このオプションは、REINPUT ステートメントの実行時にアクティブ化される端末の音声アラーム機能を提供します。 下記の「***ALARM オプション***」を参照してください。

WITH TEXT オプション

WITH TEXT は、メッセージ行に表示されるテキストを提供するために使用します。 これは通常、画面処理やエラー修正のために実行する必要がある操作を示すメッセージです。

```
[WITH] [TEXT] { * operand1 } [(attributes)] [,operand3]...7
              { operand2   }
```

オペランド定義テーブル：

オペランド	構文要素			フォーマット	ステートメント参照	ダイナミック定義
operand1	C	S		N P I B*	可	不可
operand2	C	S		A U	可	不可
operand3	C	S		A U N P I F B D T L	可	不可

* *operand1* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

operand1	**Natural** メッセージファイルからのメッセージテキスト： *operand1* は、Natural メッセージファイルから取得するメッセージテキストの番号を表します。 ユーザー定義メッセージまたは Natural システムメッセージのいずれかを取得できます。 ■ 最大 4 桁の正の値（954 など）を指定すると、ユーザー定義メッセージを取得できます。 ■ 最大 4 桁の負の値（-954 など）を指定すると、Natural システムメッセージを取得できます。 「*例 4 - WITH TEXT オプション*」も参照してください。 Natural メッセージファイルは、関連ドキュメントで説明しているように、SYSERR ユーティリティで作成およびメンテナンスします。
operand2	メッセージテキスト： *operand2* は、メッセージ行に配置されるメッセージを表します。 「*例 4 - WITH TEXT オプション*」も参照してください。
attributes	*operand1*／*operand2* にさまざまな出力属性を割り当てることができます。 これらの属性と構文については、以下の「*出力属性*」で説明します。
operand3	メッセージテキストのダイナミック置換： *operand3* は、英数字、テキスト定数、または変数名を表します。 提供された値は、*operand1* または *operand2* のいずれかで指定されるメッセージテキストの一部を置き換えるために使用されます。 表記 *:n:* は、*operand3* の内容を参照するためにメッセージテキスト内に指定します。*n* には *operand3* のオカレンス（1～7）を指定します。 「*例 4 - WITH TEXT オプション*」も参照してください。 **注意:** 複数の *operand3* を指定する場合は、それぞれをコンマで区切る必要があります。コンマを小数点文字として（セッションパラメータ DC で定義）使用し、数値定数を *operand3* として指定している場合は、コンマの前後に空白を挿入して、コンマが小数点文字と解釈されないようにします。あるいは、複数の *operand3* を INPUT 区切り文字（セッションパラメータ ID で定義）で区切ることができます。ただし、ID=/（スラッシュ）の場合は使用できません。 先頭のゼロまたは末尾の空白は、メッセージに表示される前にフィールド値から削除されます。

出力属性

attributes は、テキスト表示に使用される出力属性を示します。 可能な属性：

```
{ AD=AD-value... }
{ CD=CD-value... }...
```

指定可能なセッションパラメータ値については、『パラメータリファレンス』ドキュメントの該当するセクションを参照してください。

- 「*AD - 属性定義*」の「*フィールド表現*」
- *CD - カラー定義*

Note: コンパイラは、実際には1つの出力フィールドに複数の属性値を受け入れます。 例えば、「AD=BDI」と指定できます。 ただし、この場合は最後の値のみが適用されます。 示した例では、値 "I" のみが有効になり、出力フィールドは強調表示されます。

MARK オプション

MARK オプションを使用して、特定のフィールドをマークすることができます。つまり、REINPUT ステートメントの実行時にカーソルが配置されるフィールドを指定できます。 また、フィールド内の特定の位置をマークすることもできます。 さらに、入力保護フィールドにしたり、表示属性やカラー属性を変更できます。

```
MARK [POSITION operand4 [IN]] [FIELD] { { operand5    } [(attributes)] }...
                                      { { *fieldname } }
```

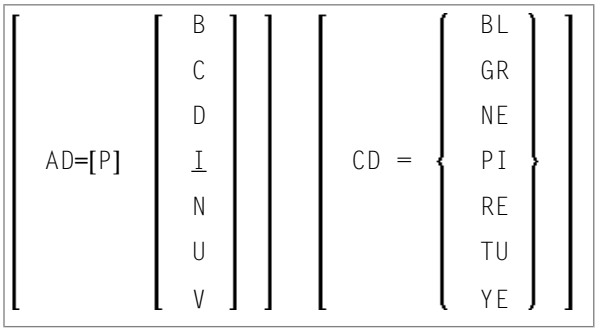
オペランド定義テーブル：

オペランド	構文要素																ステートメント参照	ダイナミック定義
operand4	C	S							N	P	I						可	不可
operand5	C	S	A						N	P	I						可	不可

構文要素の説明：

operand5	マークするフィールド： INPUT ステートメントに指定された AD=A または AD=M のフィールド（非保護フィールド）にはすべて、1 から始まるフィールド順番号が Natural によって割り当てられます。 *operand5* には、カーソルを位置づけるフィールド番号を指定します。 **fieldname*の表記は、フィールド名を参照として用いて、（INPUT ステートメントで使用した）フィールドに位置づけるために使用します。 該当する INPUT フィールドが配列の場合、配列の 1 つ以上のオカレンスを参照するために一意のインデックスまたはインデックス範囲を指定できます。 `INPUT #ARRAY (A1/1:5)` `...` `REINPUT (AD=P) 'TEXT' MARK *#ARRAY (2:3)` *operand5* も配列の場合、*operand5* の値は INPUT 配列に対するフィールド番号として使用されます。 `RESET #X(N2/1:2)` `INPUT #ARRAY ...` `...` `REINPUT (AD=P) 'TEXT' MARK #X (1:2)`
MARK POSITION	MARK POSITION では、フィールド内の特定の位置（*operand4* で指定）にカーソルを置くことができます。 「*例 3 - MARK POSITION を指定した REINPUT FULL*」も参照してください。
operand4	*operand4* ではカーソル位置を指定します。 *operand4* に小数桁を含めることはできません。
attributes	下記の「*属性割り当て*」を参照してください。

属性割り当て：



入力フィールド（AD=A または AD=M）に対しては、入力保護属性（AD=P）を指定できます。

Note: 出力専用フィールド（AD=O）として定義されたフィールドに対して入力非保護属性は指定できません。

AD=P をステートメントレベルで指定した場合、MARK オプションで指定されたフィールドを除く全フィールドが入力保護フィールドになります。

さらにフィールドの表示属性およびカラー属性を変更できます。 これらの属性の詳細については、『パラメータリファレンス』のセッションパラメータ AD および CD を参照してください。

「*例 2 - 属性を割り当てた **REINPUT***」も参照してください。

ALARM オプション

```
[AND] [SOUND] ALARM
```

このオプションは、REINPUT ステートメントの実行時にアクティブ化される端末の音声アラーム機能を提供します。 この機能を使用するには、対応するハードウェアが使用可能である必要があります。

例

例 1 - REINPUT ステートメント

```
** Example 'REIEX1': REINPUT
************************************************************************
DEFINE DATA LOCAL
1 #FUNCTION (A1)
1 #PARM     (A1)
END-DEFINE
*
INPUT #FUNCTION #PARM
*
DECIDE FOR FIRST CONDITION
  WHEN #FUNCTION = 'A' AND #PARM = 'X'
    REINPUT 'Function A with parameter X selected.'
            MARK *#PARM
  WHEN #FUNCTION = 'C' THRU 'D'
    REINPUT 'Function C or D selected.'
  WHEN #FUNCTION = 'X'
```

```
    STOP
  WHEN NONE
    REINPUT 'Please enter a valid function.'
            MARK *#FUNCTION
END-DECIDE
*
END
```

プログラム **REIEX1** の出力：

```
#FUNCTION A #PARM Y
```

Enter キーを押した後：

```

PLEASE ENTER A VALID FUNCTION
#FUNCTION A #PARM Y
```

例 2 - 属性を割り当てた REINPUT

```
** Example 'REIEX2': REINPUT  (with attributes)
************************************************************************
DEFINE DATA LOCAL
1 #A (A20)
1 #B (N7.2)
1 #C (A5)
1 #D (N3)
END-DEFINE
*
INPUT (AD=A) #A  #B  #C  #D
*
IF #A = ' ' OR #B = 0
  REINPUT (AD=P) 'RETYPE VALUES'
                 MARK *#A (AD=I CD=RE)    /* put cursor on first field
                      *#B (AD=U CD=PI)    /* and change colours
END-IF
```

```
*
END
```

例 3 - MARK POSITION を指定した REINPUT FULL

```
** Example 'REIEX3': REINPUT  (with FULL and POSITION option)
************************************************************************
DEFINE DATA LOCAL
1 #A (A20)
1 #B (N7.2)
1 #C (A5)
1 #D (N3)
END-DEFINE
*
INPUT (AD=M) #A  #B  #C  #D
*
IF #A = ' '
  COMPUTE #B = #B + #D
  RESET #D
END-IF
*
IF #A = SCAN 'TEST' OR = ' '
REINPUT FULL 'RETYPE VALUES' MARK POSITION 5 IN *#A
END-IF
*
END
```

プログラム **REIEX3** の出力：

```
RETYPE VALUES
#A                    #B        0.00 #C       #D    0
```

例 4 - WITH TEXT オプション

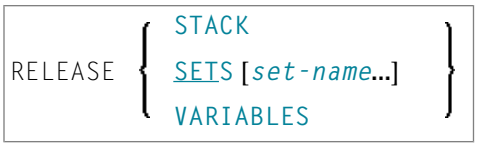
```
** Example 'REIEX4': REINPUT (with TEXT option)
************************************************************************
DEFINE DATA LOCAL
01 #NAME   (A8)
01 #TEXT   (A20)
END-DEFINE
*
*
INPUT WITH TEXT 'Enter a program name.' 'Program name:' #NAME
*
IF #NAME = ' '
  REINPUT WITH TEXT 'Input missing. Enter a name.'
END-IF
```

```
*
IF #NAME NE MASK (A)
  MOVE 'Invalid input.' TO #TEXT
  REINPUT WITH TEXT ':1: Name must start with a letter.',#TEXT
ELSE
  /* Using Natural error message 7600 for demonstration
  COMPRESS *INIT-USER 'on' *DAT4I INTO #TEXT
  INPUT WITH TEXT *-7600,#NAME,#TEXT 'Input accepted.'
END-IF
END
```

108 REJECT

このステートメントの詳細については、ACCEPT/REJECT ステートメントを参照してください。

109 RELEASE



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**STACK** | **FIND with RETAIN option** | **DEFINE DATA GLOBAL**

機能

RELEASE ステートメントは次の目的に使用します。

- Natural スタックのすべてのエントリを解放します。
- RETAIN 節を含む FIND ステートメントによって保存された ISN の集合を解放します（Adabas データベースにのみ有効）。
- グローバル変数とアプリケーション独立変数をリセットします。

構文説明

オペランド定義テーブル：

オペランド	構文要素	フォーマット	ステートメント参照	ダイナミック定義
set-name	C S	A	不可	不可

構文要素の説明：

RELEASE STACK	現在 Natural スタック内にあるすべてのデータ／コマンドを解放します。
RELEASE SETS	Adabas データベースにのみ適用できます。 *set-name*を指定せずに RELEASE SETS のみを指定すると、RETAIN 節を含む FIND ステートメントで保存されたすべての ISN 集合が解放されます。

RELEASE SETS *set-name*	特定の単一 ISN 集合を解放します。 `RELEASE SET 'CITY-SET'` `MOVE 'CITY-SET' TO #SET(A32)` `RELEASE SET #SET`
RELEASE VARIABLES	現在のグローバルデータエリアに定義されているすべての変数をそれぞれの初期値にリセットします。 また、AIV（アプリケーション独立変数）はすべて削除されるので、使用できなくなります。 これらの変数がリセット／削除されるのはレベル1のプログラムの実行終了時か、または FETCH や RUN ステートメントで他のプログラムを呼び出すときです。

例

```
** Example 'RELEX1': FIND (with RETAIN clause and RELEASE statement)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 BIRTH
  2 NAME
*
1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
     RETAIN AS 'AGESET1'
IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'
  DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
END-FIND
*
RELEASE SET 'AGESET1'
*
END
```

プログラム **RELEX1** の出力：

```
        NAME                 CITY            DATE
                                              OF
                                             BIRTH
-------------------- -------------------- ----------

RUBIN                NEW YORK             1945-10-27
WALLACE              NEW YORK             1945-08-04
```

110 REPEAT

このchapterでは、次のトピックについて説明します。

関連ステートメント：**FOR** | **ESCAPE**

関連機能グループ：「*ループ実行*」

機能

REPEAT ステートメントは、処理ループを開始するために使用します。

構文説明

このステートメントには、2 つの異なる構造が可能です。

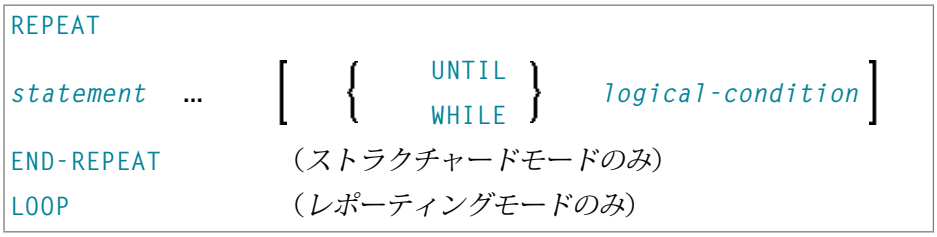
- 構文 **1** を使用すると、ステートメントは 1 回または複数回実行されます。
- 構文 **2** を使用すると、ステートメントは 0 回または複数回実行されます。

論理条件の指定位置（ループの最初または最後）によって評価のタイミングが決まります。

論理条件の詳細については、「*論理条件基準*」（『*プログラミングガイド*』）を参照してください。

構文図で使用されている記号については、「*構文記号*」を参照してください。

構文 1：



構文 2：

```
REPEAT
  [ { UNTIL } logical-condition ]      statement...
    { WHILE }
END-REPEAT          (ストラクチャードモードのみ)
LOOP                (レポーティングモードのみ)
```

構文要素の説明：

UNTIL	論理条件が真になるまで、処理ループは継続します。
WHILE	論理条件が真である限り、処理ループは継続します。
logical-condition	論理条件を指定すると、その条件によりいつループ実行を終了するかが決まります。 論理条件を指定しない場合、ループ内で ESCAPE、STOP、または TERMINATE ステートメントを指定し、ループから出る必要があります。
END-REPEAT	REPEAT ステートメントを終了するには、Natural 予約語 END-REPEAT を使用する必要があります。

例

例 1 - REPEAT

```
** Example 'RPTEX1S': REPEAT (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
*
1 #PERS-NR (A8)
END-DEFINE
*
REPEAT
  INPUT 'ENTER A PERSONNEL NUMBER:' #PERS-NR
  IF #PERS-NR = ' '
    ESCAPE BOTTOM
  END-IF
```

```
  /*
  FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PERS-NR
    IF NO RECORD FOUND
      REINPUT 'NO RECORD FOUND'
    END-NOREC
    DISPLAY NOTITLE NAME
  END-FIND
END-REPEAT
*
END
```

プログラム **RPTEX1S** の出力：

```

ENTER A PERSONNEL NUMBER: 11500304
```

人事番号を入力および確定した後：

```
        NAME
--------------------

KLUGE
```

レポーティングモードの例はライブラリ SYSEXRM のプログラム **RPTEX1R** を参照してください。

例 2 - WHILE および UNTIL オプションの使用

```
** Example 'RPTEX2S': REPEAT (with WHILE and UNTIL option)
************************************************************************
DEFINE DATA LOCAL
1 #X (I1) INIT <0>
1 #Y (I1) INIT <0>
END-DEFINE
*
REPEAT WHILE #X <= 5
  ADD 1 TO #X
  WRITE NOTITLE '=' #X
END-REPEAT
*
SKIP 3
REPEAT
  ADD 1 TO #Y
  WRITE '=' #Y
  UNTIL #Y = 6
END-REPEAT
```

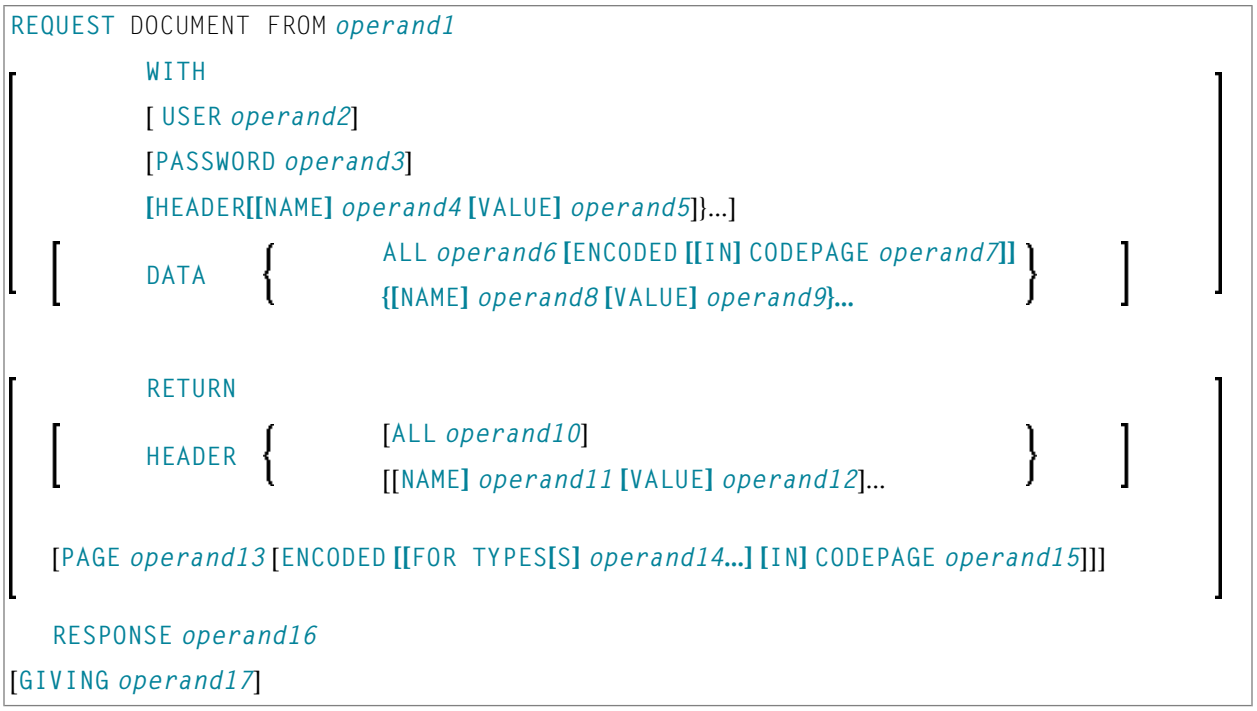
```
*
END
```

プログラム **RPTEX2S** の出力：

```
#X:     1
#X:     2
#X:     3
#X:     4
#X:     5
#X:     6



#Y:     1
#Y:     2
#Y:     3
#Y:     4
#Y:     5
#Y:     6
```

レポーティングモードの例はライブラリ SYSEXRM のプログラム **RPTEX2R** を参照してください。

111 REQUEST DOCUMENT



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

REQUEST DOCUMENT ステートメントは、外部システムにアクセスする方法を提供します。『*プログラミングガイド*』の「*インターネットと XML アクセスのステートメント*」を参照してください。

Unicode サポートの詳細については、『*Unicode およびコードページのサポート*』ドキュメントの「*ステートメント*」を参照してください。

クッキーに対する制限事項

HTTP プロトコル下では、クライアントワークステーションの状況についての情報を維持するために、サーバーはクッキーを使用します。

REQUEST DOCUMENT は、インターネットオプション設定を使用して実装されます。 これは、セキュリティ設定に依存してクッキーが使用されることを意味しています。

インターネットオプション設定 "Disabled"（使用不可）が設定されている場合、クッキーヘッダー（*operand 4/5*）が送信されても、クッキーは送信されません。

サーバー環境に対して、インターネットオプション設定 "Prompt" を使用しないでください。 クライアントはプロンプトに答えることができないため、この設定によりサーバーが「ハングアップ」 します。

メインフレーム環境ではクッキーはサポートされていないため、無視されます。

Windows 環境では、クッキーは Windows API により自動的に処理されます。 これは、ブラウザでクッキーが有効にされると、すべての受信クッキーが保存され、次の要求で自動的に送信されることを意味しています。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand1	C	S				A											不可	可
operand2	C	S				A											不可	可
operand3	C	S				A											不可	可
operand4	C	S				A											不可	可
operand5	C	S				A		N	P	I	F		D	T	L		不可	可
operand6	C	S				A	U	N	P	I	F	B	D	T	L		不可	可
operand7	C	S				A											不可	可
operand8	C	S				A											不可	可
operand9	C	S				A		N	P	I	F		D	T	L		不可	可
operand10		S				A											不可	可
operand11	C	S				A											不可	可
operand12		S				A		N	P	I	F	B	D	T	L		不可	可
operand13		S				A	U					B					不可	可
operand14	C	S				A											不可	可
operand15	C	S				A											不可	可
operand16		S								I4							不可	可
operand17		S								I4							不可	不可

構文要素の説明

DOCUMENT **FROM** *operand1*	ドキュメントの場所： *operand1* は、ドキュメントにアクセスするための URL です。 **注意:** 次の情報は、*operand1* が "http://" または "https://" で始まる場合にのみ有効です。
WITH	**WITH** 節： この節は、要求に対するオプションのユーザー／パスワード、ヘッダー、およびデータ詳細を指定するために使用します。
USER *operand2*	ユーザー名： *operand2* は、要求に対して使用されるユーザーの名前です。
PASSWORD *operand3*	ユーザーパスワード： *operand3* は、要求に対して使用されるユーザーのパスワードです。
HEADER **{[[NAME]** *operand4* **[VALUE]** *operand5*]}...	ヘッダー節： *operand4* と *operand5* は、相互に組み合わせた場合にのみ使用できます。 ■ *operand4* は、この要求によって送信される HEADER 変数の名前です。 ■ *operand5* は、この要求によって送信される HEADER 変数の値です。 **注意:** *operand4* のヘッダー名： ヘッダー名に CR/LF（改行）または ":"（コロン）を含めることはできません。 これは、REQUEST DOCUMENT ステートメントではチェックされません。 有効なヘッダー名については、HTTP 仕様を参照してください。 Web インターフェイスとの互換性のために、ヘッダー名は "-"（ダッシュ）の代わりに "_"（下線）で記述できます。 内部的に、"_" は "-" で置き換えられます。 *operand5* のヘッダー値： ヘッダー値に CR/LF を含めることはできません。 これは、REQUEST DOCUMENT ステートメントではチェックされません。 有効なヘッダー値とフォーマットについては、HTTP 仕様を参照してください。 一般情報 HTTP 要求には、いくつかのヘッダー（例：Request-Method または Content-Type）が必要です。 これらのヘッダーは、REQUEST DOCUMENT ステートメントで指定されたパラメータに応じて自動的に生成されます。 「*自動生成ヘッダー*」も参照してください。
DATA	**DATA** 節：

	特定の DATA 変数名と値（下記の*operand8* および*operand9* を参照）、または完全なドキュメント（下記の「*DATA ALL 節*」を参照）を指定できます。
ALL *operand6*	*operand6* は、送信される完全なドキュメントです。 この値は、HTTP 要求メソッド PUT で必要です（「*自動生成ヘッダー*」を参照）。 「*受信／送信データのエンコード*」の「***DATA ALL 節***」を参照してください。
[ENCODED [[IN] CODEPAGE *operand7*]	*operand6* は、現在のコードページから *operand7* で指定されたコードページにエンコードされます。 「*受信／送信データのエンコード*」の「***DATA ALL 節***」を参照してください。
{[NAME] *operand8* **[VALUE]** *operand9*}...	**DATA** 変数名と値： *operand8* と *operand9* は、相互に組み合わせた場合にのみ使用できます。 ■ *operand8* は、この要求によって送信される DATA 変数の名前です。 この値は、HTTP 要求メソッド POST で必要です（URL エンコード必須。特に "&"、"="、"%"）。 ■ *operand8* は、この要求によって送信される DATA 変数の名前です。 この値は、HTTP 要求メソッド POST で必要です（URL エンコード必須。特に "&"、"="、"%"）。 制限事項 *operand8*／*operand9*が指定され、デフォルトで通信が "http://"または "https://"の場合、コンテンツタイプ application/x-www-form-urlencoded の要求メソッド POST（「*自動生成メソッド*」を参照）が使用されます。 要求中、*operand8*／*operand9*は "=" および "&" 文字で区切られます。 したがって、"="、"&" 文字をオペランドに含めることはできません。また、URL エンコードのため "%" 文字を含めることもできません。 これらの文字は、「危険」 とみなされ、次のようにエンコードする必要があります。 **文字** / **URL エンコード構文** % / %25 & / %26 = / %3D 「***URL エンコードに対する一般的な注意***」も参照してください。
RETURN	**RETURN** 節： この節は、返される HEADER または PAGE 情報を指定するために使用します。
HEADER [ALL *operand10*]	**RETURN HEADER ALL** 節： この節を指定すると、*operand10* に HTTP レスポンスで配信されたすべてのヘッダー値が含まれます。 最初の行にはステータス情報が含まれ、すべての後続行には名前と値のペアとしてヘッダーが含まれます。 名前は常にコロン（:）で終了し、値は改行（LF）で終了します。 内部的に、すべての "CR/LF" は改行（"LF"）に変換されます。
HEADER [[NAME] *operand11*]	**RETURN HEADER NAME/VALUE** 節： この節を指定すると、特定のヘッダー情報のみが返されます。

[VALUE] *operand12*]...	*operand11* と *operand12* は、相互に組み合わせた場合にのみ使用できます。 ■ *operand11* は、この要求で受信された HEADER の名前です。 HEADER は HTTP で必要です。 ■ *operand12* は、この要求で受信された HEADER の値です。 HEADER は HTTP で必要です。 *operand11* に返されるヘッダー名： Web インターフェイスとの互換性のために、ヘッダー名は "-" の代わりに "_" で記述できます。 内部的に、"_" は "-" で置き換えられます。 *operand11* が空白の文字列の場合、ステータス情報が返されます。 `HTTP/1.0 200 OK`
RETURN PAGE	**RETURN PAGE** 節： 受信データを特定のコードページでエンコードする場合、PAGE 節を使用できます。 以下の「*受信／送信データのエンコード*」の「***RETURN PAGE** 節*」を参照してください。
PAGE *operand13*	*operand13* は、この要求に対して返されるドキュメントです。 以下の「*受信／送信データのエンコード*」の「***RETURN PAGE** 節*」を参照してください。
[ENCODED [[FOR TYPE[S] *operand14*...] **[IN] CODEPAGE** *operand15*]]	*operand14* は、*operand13* に返されるドキュメントのエンコードを実行する MIME タイプのリストです。 以下の「*受信／送信データのエンコード*」の「***RETURN PAGE** 節*」を参照してください。
	operand15 は、必要に応じて *operand13* のエンコードに使用されるコードページです。 *operator15* の値が空白の場合、変換は行われません。 *operand13* はデフォルトのコードページ（コンフィグレーションユーティリティのプロファイルパラメータ CP）でエンコードされます。 以下の「*受信／送信データのエンコード*」の「***RETURN PAGE** 節*」を参照してください。
RESPONSE *operand16*	**RESPONSE** 節： RESPONSE 節は、要求のレスポンスコード番号を表示するために使用します。 *operand16* は、要求のレスポンスコード番号です。例えば、200（要求完了）です。 「*レスポンス番号の概要 - **HTTP** 要求*」も参照してください。
GIVING *operand17*	**GIVING** 節： 要求を実行できなかった場合は、*operand17* に Natural エラーが含まれます。

自動生成ヘッダー（*operand4/5*）

Request-Method

operand5 では、HEAD、POST、GET、および PUT がサポートされています。

次の表は、指定されたオペランドに応じた Request-Method の自動的な計算を示しています。

オペランド	Request-Method			
	HEAD	**POST**	**GET**	**PUT**
WITH HEADER (*operand4*/*operand5*)	オプション	オプション	オプション	オプション
WITH DATA (*operand7*/*operand8*)	指定なし	指定	指定なし	オプション ALL（*operand6*）指定の場合のみ
RETURN HEADER (*operand10*～*operand12*)	指定	オプション	オプション	オプション
RETURN PAGE (*operand13*)	指定なし	指定	指定	オプション

Content-Type

要求メソッドが POST の場合、コンテンツタイプヘッダーを HTTP 要求で配信する必要があります。 コンテンツタイプが明示的に設定されていない場合、*operand5* に対して次の値が自動的に生成されます。

```
application/x-www-form-urlencoded
```

Note: 自動的に生成されたヘッダーを上書きすることは可能です。 Natural はエラーについてそれらをチェックしません。 予期しないエラーが起こるかもしれません。

URL エンコードに対する一般的な注意

コンテンツタイプ `application/x-www-form-urlencoded` で `POST` データを送信するとき、一定の文字が URL エンコードによって表される必要があります。これは、%16 進文字コードの文字で代用することを意味しています。 URL エンコードが必要なときとその理由の詳細については、RFC 1630、RFC 1738、RFC 1808 を参照してください。 ここでは、いくつかの基本的な詳細を示します。 すべての非 ASCII 文字（つまり、ASCII 文字でもない有効な ISO 8859/1 文字）は URL エンコードする必要があります。例えば、ファイル *köln.html* は URL で *k%F6ln.html* として表示されます。

Web ページが電子メールによって要求されるとき、いくつかの文字が 「危険」 と考えられます。

これらの文字は次のとおりです。

文字	URL エンコード構文
タブ文字	%09
スペース文字	%20
[	%5B
\	%5C
]	%5D
^	%5E
`	%60
{	%7B
|	%7C
}	%7D
~	%7E

URL を記述するとき、これらの文字の URL エンコードが必要です。

いくつかの文字は URL で特別な意味を持っています。例えば、コロン（:）は URL の残りから URL スキームを区別し、2 つのスラッシュ（//）は URL が共通インターネットスキーム構文とパーセント記号（%）に対応することを示します。 一般的に、これらの文字がファイル名の一部として出現するときには、それらを URL の特別な意味と区別するために、URL エンコードが必要です。これは単純化のためです。詳細については、RFC を参照してください。

これらの文字は次のとおりです。

文字	URL エンコード構文
"	%22
#	%23
%	%25
&	%26
+	%2B
,	%2C
/	%2F
:	%3A
<	%3C
=	%3D
>	%3E
?	%3F
@	%40

レスポンス番号の概要 - HTTP 要求

ステータス	値	レスポンス
STATUS CONTINUE	100	要求の継続は OK です
STATUS SWITCH_PROTOCOLS	101	サーバーはアップグレードヘッダーでプロトコルを切り替えました
STATUS OK	200	要求は完了しました
STATUS CREATED	201	オブジェクトが作成されました（理由 = 新しい URL）
STATUS ACCEPTED	202	非同期完了（TBS）
STATUS PARTIAL	203	部分的な完了
STATUS NO_CONTENT	204	返すための情報がありません
STATUS RESET_CONTENT	205	要求は完了しましたがフォームをクリアしてください
STATUS PARTIAL_CONTENT	206	部分的に GET を実行しました
STATUS AMBIGUOUS	300	サーバーは、何を戻すかを決定できませんでした
STATUS MOVED	301	オブジェクトは恒久的に移動しました
STATUS REDIRECT	302	オブジェクトは一時的に移動しました
STATUS REDIRECT_METHOD	303	新規アクセスメソッドを使用しないリダイレクション
STATUS NOT_MODIFIED	304	If-modified-since は修正されませんでした
STATUS USE_PROXY	305	プロキシへのリダイレクション（使用するプロキシはロケーションヘッダーで指定）
STATUS REDIRECT_KEEP_VERB	307	HTTP/1.1：同じ verb を保持してください

ステータス	値	レスポンス
STATUS BAD_REQUEST	400	無効な構文
STATUS DENIED	401	アクセスが拒否されました
STATUS PAYMENT_REQ	402	支払いが必要です
STATUS FORBIDDEN	403	要求が許可されていません
STATUS NOT_FOUND	404	オブジェクトは見つかりませんでした
STATUS BAD_METHOD	405	メソッドは許可されていません
STATUS NONE_ACCEPTABLE	406	クライアントで許容できるレスポンスが見つかりませんでした
STATUS PROXY_AUTH_REQ	407	プロキシ認証が必要です
STATUS REQUEST_TIMEOUT	408	要求待ちサーバーのタイムアウト
STATUS CONFLICT	409	ユーザーはさらに詳細な情報で再度サブミットする必要があります
STATUS GONE	410	リソースが使用できなくなっています
STATUS LENGTH_REQUIRED	411	サーバーは、長さなしの要求の受け入れを拒否しました
STATUS PRECOND_FAILED	412	要求に指定された必須条件が失敗しました
STATUS REQUEST_TOO_LARGE	413	要求エンティティが大きすぎです
STATUS URL_TOO_LONG	414	要求 URL が長すぎです
STATUS UNSUPPORTED_MEDIA	415	未サポートのメディアタイプ
STATUS SERVER_ERROR	500	内部サーバーエラー
STATUS NOT_SUPPORTED	501	"Required" はサポートされていません
STATUS BAD_GATEWAY	502	ゲートウェイからエラーレスポンスを受信しました
STATUS SERVICE_UNAVAIL	503	一時的に過負荷がかかっています
STATUS GATEWAY_TIMEOUT	504	ゲートウェイ待ちのタイムアウト
STATUS VERSION_NOT_SUP	505	HTTP バージョンがサポートされていません

レスポンス **301**～**303**（リダイレクション）

リダイレクションは、要求されたURLが移動したことを意味しています。レスポンスとして、LOCATIONという名前のReturn Headerが表示されます。このヘッダーには、要求されたページが移動したURLが含まれます。新しい`REQUEST DOCUMENT`要求は、移動したページを検索するために使用できます。

HTTPブラウザは新しいURLに自動的にリダイレクトしますが、`REQUEST DOCUMENT`ステートメントはリダイレクションを自動的に処理しません。

レスポンス **401**（拒否）

レスポンス "Access Denied" は、有効なユーザー ID とパスワードが要求で提供される場合にのみ、要求されたページにアクセスできることを意味しています。レスポンスとして、WWW-AUTHENTICATE という名前の Return Header が、この要求に必要な領域で提供されます。

HTTPブラウザは、通常、ユーザーIDとパスワードによってダイアログを表示しますが、REQUEST DOCUMENT ステートメントではダイアログは表示されません。

受信／送信データのエンコード

REQUEST DOCUMENT ステートメントを使用したデータ転送では、通常、コードページ変換は呼び出されません。 特定のコードページで受信／送信データをエンコードする場合、DATA ALL 節または RETURN PAGE 節でその指定を行います。

- DATA ALL 節
- RETURN PAGE 節

DATA ALL 節

送信データのエンコードには、DATA ALL 節を使用します。

```
ALL operand6 [ENCODED [[IN] CODEPAGE operand7]]
```

構文の説明：

ALL *operand6*	*operand6* は、送信される完全なドキュメントです。 この値は、HTTP 要求メソッド PUT で必要です（「*自動生成ヘッダー*」を参照）。
[ENCODED [[IN] CODEPAGE *operand7*]]	*operand6* は、現在のコードページから *operand7* で指定されたコードページにエンコードされます。

RETURN PAGE 節

受信データのエンコードには、RETURN PAGE 節を使用します。

```
[PAGE operand13 [ENCODED [[FOR TYPE[S] operand14...] [IN] CODEPAGE operand15]]]
```

HTTP/HTTPS 要求のレスポンスとして、受信データにバイナリデータ（例えば "image/gif"）または文字データ（例えば "text/html"）が含まれることがあります。 REQUEST DOCUMENT ステートメントは、レスポンスとともに、要求したドキュメントのコンテンツタイプ（MIME タイプ）を指定するパラメータを受け取ります。 このパラメータには、ドキュメントのエンコードに使用されるコードページに関する情報が含まれることがあります。

この節では、Natural の現在のコードページへの自動変換を行います。

構文の説明：

RETURN PAGE *operand13*	返されるページに対してエンコードは行われません。つまり、ページのエンコードはHTTPサーバーから配信されたときの状態のまま変わりません。
RETURN PAGE *operand13* **ENCODED**	返される MIME タイプにエンコードが含まれる場合、*operand13* はこのコードページから現在のコードページ（A/B）または（U）にエンコードされます。 下記の注を参照してください。
RETURN PAGE *operand13* **ENCODED [IN] CODEPAGE** *operand15*	返される MIME タイプにエンコードが含まれない場合、*operand13* は *operand15* で定義されたコードページから現在のコードページ（A/B）または（U）にエンコードされます。
RETURN PAGE *operand13* **[ENCODED [[FOR TYPE[S]** *operand14...]* **[IN] CODEPAGE** *operand15***]]**	返される MIME タイプにエンコードが含まれない場合、その MIME タイプが*operand14* で指定されたタイプのいずれかと一致しているかどうか、追加のチェックが行われます。一致している場合、*operand13*は *operand15* で定義されたコードページから現在のコードページ（A/B）または（U）にエンコードされます。

Note: 「返される MIME タイプにエンコードが含まれる」 ということは、HTTP サーバーから `charset=`節（例：`charset=ISO-8859-1`）を含むコンテンツタイプヘッダーが返されることを意味します。

例

- 例 1 - 一般的な要求
- 例 2 - 単純な Get 要求（データなし）
- 例 3 - 単純な Head 要求（戻りページなし）
- 例 4 - 単純な Post 要求（デフォルト）
- 例 5 - 単純な Put 要求（すべてのデータを処理）

Note: このステートメントのダイアログ例 V5-RDOC がライブラリ SYSEXV にあります。

例 1 - 一般的な要求

```
REQUEST DOCUMENT FROM "http://bolsap1:5555/invoke/sap.demo/handle_RFC_XML_POST"
  WITH
   USER #User PASSWORD #Password
   DATA
   NAME 'XMLData'        VALUE #Queryxml
   NAME 'repServerName' VALUE 'NT2'
  RETURN
```

```
   PAGE #Resultxml
  RESPONSE #rc
```

例 2 - 単純な Get 要求（データなし）

```
REQUEST DOCUMENT FROM "http://pcnatweb:8080"
  RETURN
   PAGE #Resultxml
  RESPONSE #rc
```

例 3 - 単純な Head 要求（戻りページなし）

```
REQUEST DOCUMENT FROM "http://pcnatweb"
   RESPONSE #rc
```

例 4 - 単純な Post 要求（デフォルト）

```
REQUEST DOCUMENT FROM "http://pcnatweb/cgi-bin/nwwcgi.exe/sysweb/nat-env"
  WITH
   DATA
   NAME 'XMLData'       VALUE #Queryxml
   NAME 'repServerName' VALUE 'NT2'
  RETURN
   PAGE #Resultxml
  RESPONSE #rc
```

例 5 - 単純な Put 要求（すべてのデータを処理）

```
REQUEST DOCUMENT FROM "http://pcnatweb/test.txt"
  WITH
   DATA ALL     #document
  RETURN
   PAGE #Resultxml
  RESPONSE #rc
```

112 RESET

```
RESET [INITIAL] operand1 ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ADD** | **COMPRESS** | **COMPUTE** | **DIVIDE** | **EXAMINE** | **MOVE** | **MOVE ALL** | **MULTIPLY** | **SEPARATE** | **SUBTRACT**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

RESET ステートメントは、フィールドの値をリセットするために使用します。

- RESET（INITIAL なし）は、指定された各フィールドの内容をフォーマットに依存したデフォルトの初期値に設定します。
- RESET INITIAL は、指定された各フィールドを DEFINE DATA ステートメントのフィールド定義に従った初期値に設定します。 DEFINE DATA ステートメントで INIT 節を指定せずに宣言したフィールドに対して、RESET INITIAL は RESET（INITIAL なし）と同じ効果を持ちます。

Notes:

1. DEFINE DATA ステートメントで CONSTANT 節を指定して宣言したフィールドは、内容を変更できないので、RESET ステートメントで参照できません。
2. レポーティングモードでは、プログラムに DEFINE DATA LOCAL ステートメントが含まれていなければ、RESET ステートメントを使用して変数を定義することもできます。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1		S	A	G	M	A	U	N	P	I	F	B	D	T	L	C	G	O	可	可

構文要素の説明：

RESET *operand1*	空値にリセット： RESET（INITIAL なし）は、指定された各フィールドの内容（*operand1*）をデフォルトの初期値にリセットします。 *operand1* がダイナミック変数の場合、RESET ステートメントの実行時に変数に含まれている長さの空値にリセットされます。ダイナミック変数の現在の長さは、システム変数*LENGTH を使用して確認できます。 ダイナミック変数に関する全般的な情報については、「*ラージ変数／フィールドとダイナミック変数／フィールド*」を参照してください。
RESET INITIAL *operand1*	初期値にリセット： RESET INITIAL は、指定された各フィールド（*operand1*）を DEFINE DATA ステートメントのフィールド定義に従った初期値に設定します。 ■ DEFINE DATA ステートメントに INIT 値を指定していない場合、フィールドはそのフォーマットに基づいてデフォルトの初期値で初期化されます。 ■ ダイナミック変数を使用する場合、初期値が定義されていなければ、*LENGTH はゼロに設定されます。 ■ RESET INITIAL を配列に適用する場合、DEFINE DATA ステートメントで定義した配列全体に適用する必要があります。配列の個々のオカレンスに RESET INITIAL を適用することはできません。 ■ X-array を使用する場合、*OCCURRENCE がゼロに設定されます。 ■ 再定義で発生したフィールドに RESET INITIAL は使用できません。 ■ RESET INITIAL はダイナミック変数に適用されます。 ■ RESET INITIAL はデータベースフィールドには適用できません。

例

```
** Example 'RSTEX1': RESET (with/without INITIAL)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 #BINARY  (B4) INIT <1>
1 #INTEGER (I4) INIT <5>
1 #NUMERIC (N2) INIT <25>
END-DEFINE
*
LIMIT 1
```

```
READ EMPLOY-VIEW
  /*
  WRITE NOTITLE 'VALUES BEFORE RESET STATEMENT:'
  WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
  /*
  RESET NAME #BINARY #INTEGER #NUMERIC
  /*
  WRITE /// 'VALUES AFTER  RESET STATEMENT:'
  WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
  /*
  RESET INITIAL #BINARY #INTEGER #NUMERIC
  /*
  WRITE /// 'VALUES AFTER  RESET INITIAL STATEMENT:'
  WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
  /*
END-READ
END
```

プログラム **RSTEX1** の出力：

```
VALUES BEFORE RESET STATEMENT:

NAME: ADAM                #BINARY: 00000001 #INTEGER:           5 #NUMERIC:
 25



VALUES AFTER  RESET STATEMENT:

NAME:                     #BINARY: 00000000 #INTEGER:           0 #NUMERIC:
  0



VALUES AFTER  RESET INITIAL STATEMENT:

NAME:                     #BINARY: 00000001 #INTEGER:           5 #NUMERIC:
 25
```

113 RESIZE

```
RESIZE { dynamic-clause } [GIVING operand5]
       { array-clause   }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**EXPAND** | **REDUCE**

関連機能グループ：「*ダイナミック変数または X-array のメモリ管理制御*」

機能

RESIZE ステートメントは、次を調整するために使用します。

- ダイナミック変数（*dynamic-clause*）のサイズ、または
- X-array（*array-clause*）のオカレンス数

詳細については、『*プログラミングガイド*』の次のセクションを参照してください。

ダイナミック変数の使用
ダイナミック変数のメモリスペースの割り当て／解放
X-array
X-Group 配列のストレージ管理

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1		S	A			A	U					B							不可	不可
operand2	C	S								I									不可	不可
operand3			A	G		A		N	P	I	F	B	D	T	L	C	G	O	可	不可
operand4	C	S						N	P	I									不可	不可
operand5		S								I4									不可	可

構文要素の説明：

dynamic-clause	RESIZE DYNAMIC ステートメントは、ダイナミック変数（*operand1*）の現在割り当てられているストレージについて、割り当てられている長さを、*operand2*で指定した値に調整します。 詳細については、後述の「***Dynamic** 節*」を参照してください。
operand1	*operand1* は、長さが調整されるダイナミック変数です。
operand2	*operand2*は、ダイナミック変数の新しい長さを指定するために使用します。 値として、負ではない整数の定数またはInteger4（I4）タイプの変数を指定する必要があります。
array-clause	RESIZE ARRAY ステートメントは、X-array（*operand3*）のオカレンス数を、*(dim[,dim[,dim]])* で指定した上下限に調整します。 詳細については、以下の「***Array** 節*」を参照してください。
operand3	*operand3* は X-array です。 X-array のオカレンスを拡張または削減できます。 配列のインデックス表記はオプションです。 各次元でインデックス表記として使用できるのは、全範囲を示す表記 "*" のみです。
dim *operand4*	X-array 拡張の上下限表記（*operand4* またはアスタリスク）は、ここで指定します。 現在の上下限の値を使用する必要がある場合は、*operand4*の代わりにアスタリスク（*）を指定する必要があります。 詳細については、後述の「*次元*」を参照してください。
GIVING *operand5*	GIVING 節を指定しない場合は、エラー発生時に Natural ランタイムエラー処理がトリガされます。 GIVING 節を指定した場合は、*operand5* に、エラー発生時は Natural メッセージ番号、成功時はゼロが含まれます。

Dynamic 節

```
[SIZE OF] DYNAMIC [VARIABLE] operand1 TO operand2
```

RESIZE DYNAMIC ステートメントは、ダイナミック変数（*operand1*）の割り当てられている長さを、*operand2* で指定した値に調整します。

RESIZE ステートメントを使用した場合、現在割り当てられているストレージサイズが、増減のどちらが必要であるかに関係なく、要求された値に調整されます。

Array 節

```
[AND RESET] [OCCURRENCES OF] ARRAY operand3 TO (dim[,dim[,dim]])
```

RESIZE ARRAY ステートメントは、X-array（*operand3*）のオカレンス数を、(*dim*[,*dim*[,*dim*]]) で指定した上下限に調整します。

RESET オプションは、サイズ変更した X-array のすべてのオカレンスをデフォルトのゼロ値にリセットします。 デフォルト（RESET オプションなし）では、実際の値は保持され、サイズ変更した（新しい）オカレンスがリセットされます。

RESIZE ステートメントで使用される上限または下限は、配列に定義された対応する上限または下限と正確に同じである必要があります。

例：

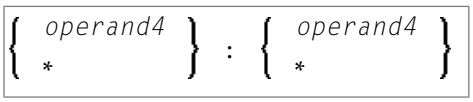
```
DEFINE DATA LOCAL
1 #a(I4/1:*)
1 #g(1:*)
  2 #ga(I4/1:*)

1 #i(i4)
END-DEFINE
...

*/ resizing #a (1:10)
RESIZE ARRAY #a TO (1:10)       /* #a is resized to
RESIZE ARRAY #a TO (*:10)       /* 10 occurrences.


/* resizing #ga (1:10,1:20)
RESIZE ARRAY #g TO (1:10)       /* 1st dimension is set to (1:10)
RESIZE ARRAY #ga TO (*:*,1:20)  /* 1st dimension is dependent and
                                /* therefore kept with (*:*)
                                /* 2nd dimension is set to (1:20)

RESIZE ARRAY #a TO (5:10)       /* This is rejected because the lower index
                                /* must be 1 or *
RESIZE ARRAY #a TO (#i:10)      /* This is rejected because the lower index
                                /* must be 1 or *

RESIZE ARRAY #ga TO (1:10,1:20) /* (1:10) for the 1st dimension is rejected
                                /* because the dimension is dependent and
                                /* must be specified with (*:*).
```

詳細については、次を参照してください。

■ *X-array のストレージ管理*

■ *X-Group 配列のストレージ管理*

次元



X-array 拡張の上下限表記（*operand4* またはアスタリスク）は、ここで指定します。 現在の上下限の値を使用する必要がある場合は、*operand4* の代わりにアスタリスク（*）を指定できます。 "*.*" の代わりに単一のアスタリスクを指定できます。

次元数（*dim*）は、X-array（1、2、または 3）と正確に一致している必要があります。

指定した次元のオカレンス数が、現在割り当てられているオカレンス数より小さい場合は、対応する次元のオカレンス数は変更されません。

114 RETRY

RETRY

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **STORE** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

RETRY ステートメントは、ON ERROR ステートメントブロック内でのみ使用できます（ON ERROR ステートメントを参照）。 他ユーザー用にホールド状態になっているレコードに再度アクセスするときに使用します。

レコードが他ユーザー用にすでにホールド状態になっている場合、Naturalはエラーメッセージ3145を発行します。 セッションパラメータ WH（ホールド状態でのレコードの待機）も参照してください。

RETRY ステートメントは、エラー 3145 の原因となっているオブジェクト内に指定します。

レコードのホールドロジックの詳細については、『*プログラミングガイド*』の「*レコードホールドロジック*」セクションを参照してください。

制限事項

このステートメントは、Adabas データベースにアクセスするためにのみ使用できます。

例

```
** Example 'RTYEX1': RETRY
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
*
```

```
1 #RETRY (A1) INIT <' '>
END-DEFINE
*
FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
  /*
  DELETE
  END TRANSACTION
  /*
  ON ERROR
    IF *ERROR-NR = 3145
      INPUT NO ERASE 10/1
            'RECORD IS IN HOLD' /
            'DO YOU WISH TO RETRY?' /
            #RETRY '(Y)ES OR (N)O?'
      IF #RETRY = 'Y'
        RETRY
      ELSE
        STOP
      END-IF
    END-IF
  END-ERROR
  /*
  AT END OF DATA
    WRITE NOTITLE *NUMBER 'RECORDS DELETED'
  END-ENDDATA
END-FIND
*
END
```

115 RUN

```
RUN [REPEAT] operand1 [ operand2 [(parameter)]] ... 40
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

RUN ステートメントは、Natural システムファイルから Natural ソースプログラムを読み込んで実行するときに使用します。

Natural RPC については、『*Natural リモートプロシージャコール（RPC）*』ドキュメントの「*サーバーに対する Natural ステートメントの注意事項*」を参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素			フォーマット	ステートメント参照	ダイナミック定義
operand1	C	S		A	可	不可
operand2	C	S	A	G A U N P I F B D T L G	可	不可

構文要素の説明：

REPEAT	RUN REPEAT は、複数の出力画面が（INPUT ステートメントによって）生成されるような場合でも、プログラムの実行が終了するまで、ユーザーに入力を促さないようにします。 この機能は、各画面でユーザーが応答せずにプログラムで複数画面の情報を表示するような場合に使用できます。
operand1	プログラム名： *operand1* として、プログラムの名前を英数字定数または英数字変数の内容として指定できます。 変数を使用する場合は、長さは 8 文字長で指定する必要があります。 プログラムは、現在のライブラリまたは連結されたライブラリ（デフォルトの STEPLIB は SYSTEM）に保存できます。プログラムが見つからない場合、エラーメッセージが発行されます。 プログラムはソースプログラムワークエリア内に読み込まれ、エリア内にあるソースプログラムに置き換わります。
operand2	パラメータ：

	RUN ステートメントは、実行するプログラムに対してパラメータを渡すためにも使用できます。 パラメータは任意のフォーマットで指定できます。 パラメータは、対応する INPUT フィールドに適したフォーマットに変換されます。 すべてのパラメータは、Natural スタックの最上位に配置されます。 パラメータは INPUT ステートメントで読み込まれます。 最初の INPUT ステートメントが発行されると、INPUT ステートメントで指定されたフィールドに全パラメータが挿入されます。 INPUT ステートメントには数字フォーマットで定義されたパラメータフィールドに対する符号指定（SG=ON）が必要です。 次の INPUT ステートメントで読み取られる以上のパラメータが渡されると、余分のパラメータは無視されます。 パラメータの数はシステム変数 *DATA で取得できます。 **注意:** *operand2* が時刻変数（フォーマット T）の場合は、変数内容のうち時刻コンポーネントのみが渡され、日付コンポーネントは渡されません。
parameter	*operand2* が日付変数の場合は、この変数に対する *parameter* として、セッションパラメータ DF（『パラメータリファレンス』を参照）を指定できます。

ダイナミックなソーステキストの作成／実行

ソースまたはその一部をダイナミックに作成したプログラムは、RUN ステートメントでダイナミックにコンパイルおよび実行できます。

ダイナミックなソーステキスト作成を行うには、グローバル変数にソーステキストを挿入し、その後、ソーステキストでそのグローバル変数を参照します。そのとき、変数名の先頭文字の "+" を "&" に置き換えて参照します。 グローバル変数の内容は、プログラムが RUN ステートメントで呼び出されたときに、ソーステキストとして解釈されます。

インデックス付きグローバル変数を、RUN ステートメントを介して呼び出されるプログラム内で使用しないでください。

グローバル変数にコメントや INCLUDE ステートメントを入れることはできません。

例

RUN ステートメントを含むプログラム：

```
** Example 'RUNEX1': RUN (with dynamic source program creation)
************************************************************************
DEFINE DATA
GLOBAL
  USING RUNEXGDA
LOCAL
1 #NAME (A20)
1 #CITY (A20)
END-DEFINE
*
INPUT 'Please specify the search values:' //
      'Name:' #NAME /
      'City:' #CITY
*
RESET +CRITERIA     /* defined in GDA 'RUNEXGDA'
*
IF #NAME = ' ' AND #CITY = ' '
  REINPUT 'Enter at least 1 value'
END-IF
*
IF #NAME NE ' '
  COMPRESS 'NAME' ' =''' #NAME '''' INTO +CRITERIA LEAVING NO
END-IF
IF #CITY NE ' '
  IF +CRITERIA NE ' '
    COMPRESS +CRITERIA 'AND' INTO +CRITERIA
  END-IF
  COMPRESS +CRITERIA ' CITY =''' #CITY '''' INTO +CRITERIA LEAVING NO
END-IF
*
RUN 'RUNEXFND'
*
END
```

RUN ステートメントで実行されるプログラム **RUNEXFND**：

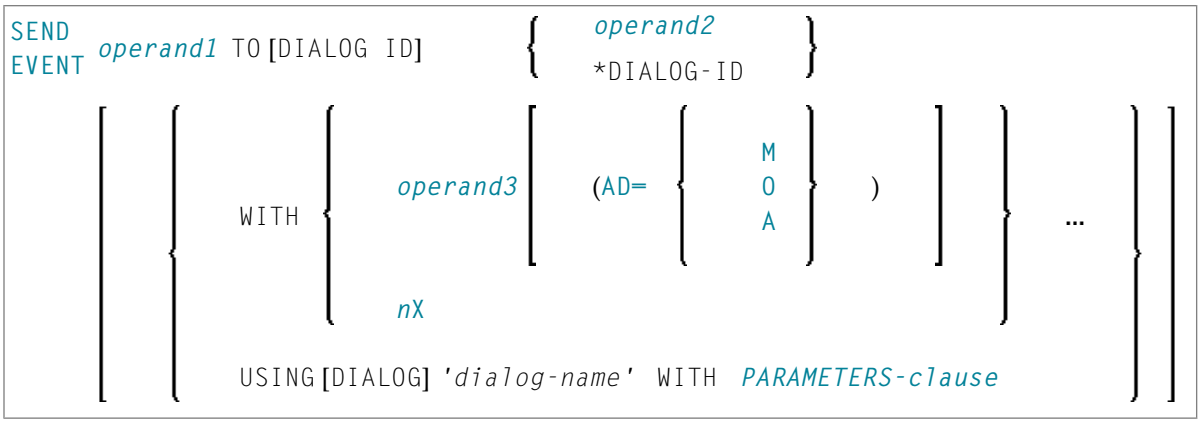
```
** Example 'RUNEXFND': RUN (program executed with RUN in RUNEX1)
************************************************************************
DEFINE DATA
GLOBAL
  USING RUNEXGDA
LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
END-DEFINE
*
* &CRITERIA filled with  "NAME = 'xxxxx' AND CITY = 'xxxx'"
*
FIND NUMBER EMPLOY-VIEW WITH &CRITERIA
```

```
     RETAIN AS 'EMP-SET'
DISPLAY *NUMBER
*
END
```

グローバルデータエリア **RUNEXGDA**：

```
Global    RUNEXGDA  Library SYSEXSYN                      DBID    10 FNR    32
Command                                                                    > +
I T L  Name                             F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->
     1 +CRITERIA                       A         80
```

116 SEND EVENT



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**OPEN DIALOG** | **CLOSE DIALOG** | **PROCESS GUI**

関連機能グループ：「*イベントドリブンプログラミング*」

機能

SEND EVENT ステートメントは、Natural アプリケーション内でユーザー定義イベントを起動するために使用します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1	C	S				A													可	不可
operand2		S								I									可	不可
operand3	C	S	A			A		N	P	I	F	B	D	T	L	C	G	O	可	不可

構文要素の説明：

operand1 *operand2*	オペランド： *operand1* は、送信されるイベントの名前です。 *operand2*は、ユーザーイベントを受け取るダイアログの識別子です。 *operand2*は、フォーマット／長さ I4 で定義する必要があります。	
operand3	ダイアログボックスへのパラメータの引き渡し： ダイアログにパラメータを渡すことができます。 *operand3* として、ダイアログに渡されるパラメータを指定します。 *PARAMETERS-clause* を使用して、パラメータを選択的に渡すことができます。 下記の「***PARAMETERS 節***」を参照してください。	
AD=	属性割り当て： *operand3* が変数の場合は、次のいずれかの方法でマークすることができます。	
	AD=O	変更不可。セッションパラメータ AD=O を参照してください。
	AD=M	変更可。セッションパラメータ AD=M を参照してください。 これはデフォルト設定です。
	AD=A	入力のみ。セッションパラメータ AD=A を参照してください。
	operand3 が定数の場合は、 *operand3* を明示的に指定することはできません。 定数には常に AD=O が適用されます。	
***n*X**	省略されるパラメータの指定： 表記 *n*X を使用して、次の *n* パラメータを省略するように指定できます。例えば、1X は次のパラメータを省略し、3X は次の 3 つのパラメータを省略します。これは、次の *n* パラメータに、ダイアログボックスに渡す値がないことを意味します。 省略されるパラメータは、ダイアログボックスの DEFINE DATA PARAMETER ステートメントでキーワード OPTIONAL を使用して定義する必要があります。 OPTIONAL は、値を呼び出し側オブジェクトからこのようなパラメータに渡すこともできるということを意味します。	

PARAMETERS 節

```
PARAMETERS {parameter-name=operand3} ...
END-PARAMETERS
```

Note: 指定したターゲットダイアログ（*dialog-name*）がカタログされている場合にのみ、PARAMETERS 節を使用できます。

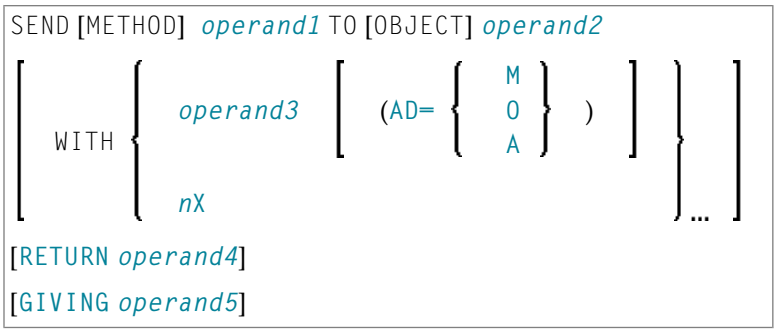
dialog-name は、ユーザーイベントを受け取るダイアログの名前です。

Note: AD=0でマークされ「参照によって」渡されたパラメータの値がダイアログボックス内で変更されると、ランタイムエラーが発生します。

詳細と例

『*プログラミングガイド*』の「*イベントドリブンプログラミング手法*」を参照してください。

117 SEND METHOD



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**CREATE OBJECT** | **DEFINE CLASS** | **INTERFACE** | **METHOD** | **PROPERTY**

関連機能グループ：「*コンポーネントベースプログラミング*」

機能

SEND METHOD ステートメントは、オブジェクトの特定メソッドを呼び出すために使用します。コンポーネントベースプログラミングの詳細については、『*プログラミングガイド*』の「*NaturalX*」セクションを参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1	C	S				A													可	不可
operand2		S																O	不可	不可
operand3	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	O	可	不可
operand4		S	A			A	U	N	P	I	F	B	D	T	L	C	G	O	可	不可
operand5		S			N					I									可	不可

ローカルクラスのメソッドに渡すことができるのは、フォーマットCおよびGのみです。詳細については、「*ローカルクラス*」セクションを参照してください。

構文要素の説明：

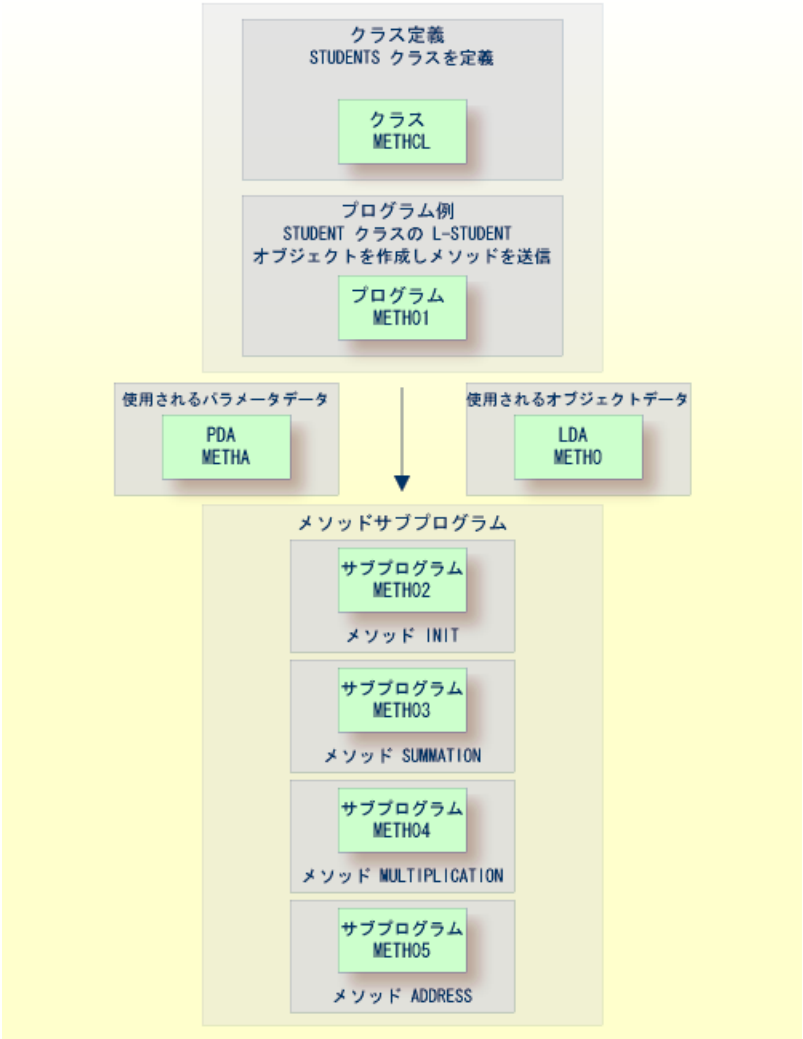
operand1	メソッド名： *operand1*は、*operand2*で指定されるオブジェクトによってサポートされるメソッドの名前です。 メソッド名はクラスの異なるインターフェイスで同一にできるため、混同を避けるために、*operand1*のメソッド名をインターフェイス名で修飾することもできます。 次の例では、オブジェクト#O3は、メソッドStart付きのインターフェイスIterateを持っています。 次のステートメントが適用されます。 `* Specifying only the method name.` `SEND 'Start' TO #O3` `* Qualifying the method name with the interface name.` `SEND 'Iterate.Start' TO #O3` インターフェイス名が指定されていない場合、Naturalはクラスのすべてのインターフェイスのメソッド名を検索します。 メソッド名が複数のインターフェイスで見つかった場合、ランタイムエラーが発生します。
operand2	オブジェクトハンドル： メソッドコールが送信されるオブジェクトのハンドル。 *operand2*は、オブジェクトハンドル（HANDLE OF OBJECT）として定義する必要があります。 オブジェクトがすでに存在している必要があります。 メソッド内で現在のオブジェクトのメソッドを呼び出すには、システム変数*THIS-OBJECTを使用してください。
operand3	メソッドに固有のパラメータ： *operand3*として、メソッドに固有のパラメータを指定できます。 次の例では、オブジェクト#O3は、パラメータPos付きのメソッドPositionToを持っています。 メソッドは次の方法で呼び出されます。 `SEND 'PositionTo' TO #O3 WITH Pos` メソッドはオプションのパラメータを持つことができます。メソッドが呼び出される場合は、オプションのパラメータを指定する必要はありません。 オプションパラメータを省略するには、プレースホルダ1Xを使用します。 *n*オプションパラメータを省略するには、プレースホルダ*n*Xを使用します。

	次の例では、オブジェクト #O4 のメソッド SetAddress はパラメータ FirstName、MiddleInitial、LastName、Street、City を持ち、そのうち MiddleInitial、Street、City がオプションです。 次のステートメントが適用されます。 ``` * Specifying all parameters. SEND 'SetAddress' TO #O4 WITH FirstName MiddleInitial LastName Street City * Omitting one optional parameter. SEND 'SetAddress' TO #O4 WITH FirstName 1X LastName Street City * Omitting all optional parameters. SEND 'SetAddress' TO #O4 WITH FirstName 1X LastName 2X ``` オプションでない（必須）パラメータを省略すると、ランタイムエラーが発生します。
AD=	属性定義： *operand3* が変数の場合は、次のいずれかの方法でマークすることができます。
	AD=O：変更不可。セッションパラメータ AD=O を参照してください。
	AD=M：変更可。セッションパラメータ AD=M を参照してください。 これはデフォルト設定です。
	AD=A：入力のみ。セッションパラメータ AD=A を参照してください。
	operand3 が定数の場合は、AD を明示的に指定することはできません。 定数には常に AD=O が適用されます。
***n*X**	省略されるパラメータ： 表記 *n*X を使用して、次の *n* パラメータを省略するように指定できます。例えば、1X は次のパラメータを省略し、3X は次の 3 つのパラメータを省略します。 これは、次の n パラメータに、メソッドに渡す値がないことを意味します。 Natural で実装されたメソッドについて、省略されるパラメータは、ダイアログの DEFINE DATA PARAMETER ステートメントでキーワード OPTIONAL を使用して定義する必要があります。 OPTIONAL は、値を呼び出し側オブジェクトからこのようなパラメータに渡すこともできるということを意味します。
RETURN *operand4*	**RETURN** 節： RETURN 節を省略し、メソッドが戻り値を持っている場合、戻り値は破棄されます。 RETURN 節を指定した場合、*operand4* にはメソッドの戻り値が含まれます。 メソッドの実行が失敗すると、*operand4* はその初期値にリセットされます。 **注意:** Natural で記述されたクラスについて、メソッドの戻り値は、メソッドのパラメータデータエリアに 1 つの追加パラメータを入力し、それを BY VALUE RESULT でマークすることによって定義されます（詳細については、「*PARAMETER 節*」セクションを参照）。したがって、Natural で記述され、戻り値を持つメソッドのパラメータデータエリアには常にメソッドパラメータの次に 1 つの追加フィールドが含まれます。 この点は、Natural で記述さ

	れたクラスのメソッドを呼び出し、SEND ステートメントのメソッドのパラメータデータエリアを使用したいときに考慮します。
GIVING *operand5*	**GIVING 節：** GIVING 節を指定しなかった場合、エラーの発生時に Natural ランタイムエラー処理が開始されます。 GIVING 節を指定した場合、*operand5* に、エラーの発生時は Natural メッセージ番号、成功時はゼロが含まれます。

例

次の図は、この例で使用されているプログラミングオブジェクトの概要を示しています。 対応するソースコードとプログラム出力を下記に示します。



プログラム **METH01 -** クラスと複数のメソッドを使用した **CREATE OBJECT** および **SEND METHOD**

```
** Example 'METH01':  CREATE OBJECT and SEND METHOD
**                     using a class and several methods (see METH*)
************************************************************************
DEFINE DATA
LOCAL
  USING METHA
LOCAL
1 L-STUDENT HANDLE OF OBJECT
1 #NAME     (A20)
1 #STREET   (A20)
1 #CITY     (A20)
1 #SUM      (I4)
1 #MULTI    (I4)
END-DEFINE
*
CREATE OBJECT L-STUDENT OF CLASS 'STUDENTS' /* see METHCL for class
*
L-STUDENT.<> := 'John Smith'
*
SEND METHOD 'INIT' TO L-STUDENT             /* see METHCL
     WITH #VAR1 #VAR2 #VAR3 #VAR4
*
SEND METHOD 'SUMMATION' TO L-STUDENT        /* see METHCL
     WITH #VAR1 #VAR2 #VAR3 #VAR4
*
SEND METHOD 'MULTIPLICATION' TO L-STUDENT   /* see METHCL
     WITH #VAR1 #VAR2 #VAR3 #VAR4
*
#NAME  := L-STUDENT.<>
#SUM   := L-STUDENT.<>
#MULTI := L-STUDENT.<>
*
SEND METHOD 'ADDRESS' TO L-STUDENT          /* see METHCL
*
#STREET := L-STUDENT.<>
#CITY   := L-STUDENT.<>
*
*
WRITE 'Name  :' #NAME
WRITE 'Street:' #STREET
WRITE 'City  :' #CITY
WRITE ' '
WRITE 'The summation of     ' #VAR1 #VAR2 #VAR3 #VAR4
WRITE 'is' #SUM
WRITE 'The multiplication of' #VAR1 #VAR2 #VAR3 #VAR4
WRITE 'is' #MULTI
```

```
*
END
```

METH01 で使用されるクラス定義 **METHCL**：

```
** Example 'METHCL': DEFINE CLASS (used by METH01)
************************************************************************
* Defining class STUDENTS for METH01
*
DEFINE CLASS STUDENTS
  OBJECT
    USING METHO                  /* Object data for class STUDENTS
  /*
  INTERFACE STUDENT-ARITHMETICS
    PROPERTY FULL-NAME
      IS NAME
    END-PROPERTY
    PROPERTY SUM
    END-PROPERTY
    PROPERTY MULTI
    END-PROPERTY
*
    METHOD INIT
      IS METH02
      PARAMETER USING METHA
    END-METHOD
    METHOD SUMMATION
      IS METH03
      PARAMETER USING METHA
    END-METHOD
    METHOD MULTIPLICATION
      IS METH04
      PARAMETER USING METHA
    END-METHOD
  END-INTERFACE
*
  INTERFACE STUDENT-ADDRESS
    PROPERTY STUDENT-NAME
      IS NAME
    END-PROPERTY
    PROPERTY STREET
    END-PROPERTY
    PROPERTY CITY
    END-PROPERTY
*
    METHOD ADDRESS
      IS METH05
    END-METHOD
  END-INTERFACE
```

```
END-CLASS
END
```

クラス **METHCL** およびサブプログラム **METH02**、**METH03**、**METH04**、**METH05** で使用されるローカルデータエリア **METHO**（オブジェクトデータ）：

```
Local     METHO     Library SYSEXSYN                      DBID    10 FNR    32
Command                                                                  > +
I T L  Name                            F Length     Miscellaneous
All -- --------------------------------- - ---------- ------------------------->
     1 NAME                            A         20
     1 STREET                          A         30
     1 CITY                            A         20
     1 SUM                             I          4
     1 MULTI                           I          4
```

プログラム **METH01**、クラス **METHCL**、およびサブプログラム **METH02**、**METH03**、**METH04** で使用されるパラメータデータエリア **METHA**：

```
Parameter METHA     Library SYSEXSYN                      DBID    10 FNR    32
Command                                                                  > +
I T L  Name                            F Length     Miscellaneous
All -- --------------------------------- - ---------- ------------------------->
     1 #VAR1                           I          4
     1 #VAR2                           I          4
     1 #VAR3                           I          4
     1 #VAR4                           I          4
```

サブプログラム **METH02 -** プログラム **METH01** で使用されるメソッド **INIT**：

```
** Example 'METH02': Method INIT (used by METH01)
************************************************************************
DEFINE DATA
PARAMETER
  USING METHA
OBJECT
  USING METHO
END-DEFINE
*
* Method INIT of class STUDENTS
*
#VAR1 := 1
#VAR2 := 2
#VAR3 := 3
#VAR4 := 4
```

```
*
END
```

サブプログラム **METH03 -** プログラム **METH01** で使用されるメソッド **SUMMATION**：

```
** Example 'METH03': Method SUMMATION (used by METH01)
************************************************************************
DEFINE DATA
PARAMETER
  USING METHA
OBJECT
  USING METHO
END-DEFINE
*
* Method SUMMATION of class STUDENTS
*
COMPUTE SUM = #VAR1 + #VAR2 + #VAR3 + #VAR4
END
```

サブプログラム **METH04 -** プログラム **METH01** で使用されるメソッド **MULTIPLICATION**：

```
** Example 'METH04': Method MULTIPLICATION (used by METH01)
************************************************************************
DEFINE DATA
PARAMETER
  USING METHA
OBJECT
  USING METHO
END-DEFINE
*
* Method MULTIPLICATION of class STUDENTS
*
COMPUTE MULTI = #VAR1 * #VAR2 * #VAR3 * #VAR4
END
```

サブプログラム **METH05 -** プログラム **METH01** で使用されるメソッド **ADDRESS**：

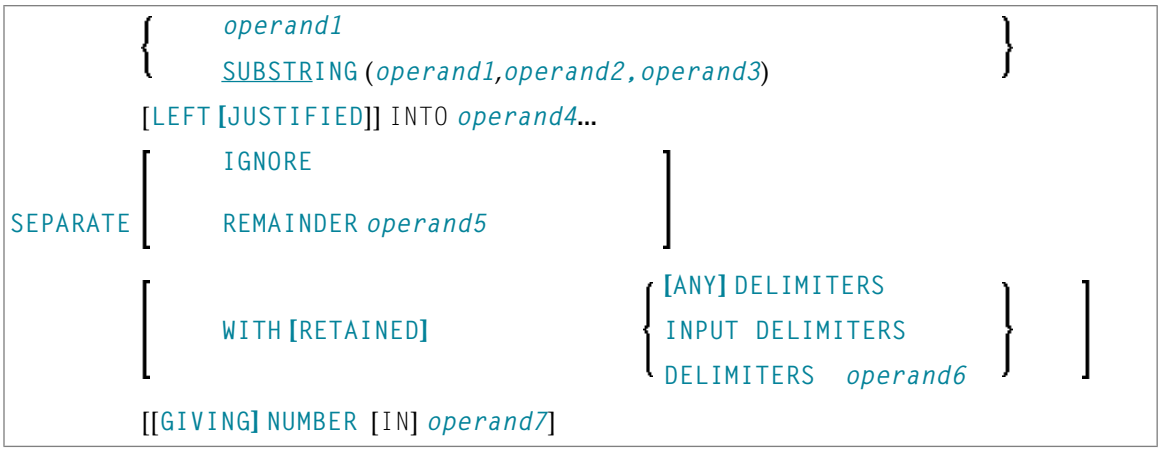
```
** Example 'METH05': Method ADDRESS (used by METH01)
************************************************************************
DEFINE DATA
  OBJECT USING METHO
END-DEFINE
*
* Method ADDRESS of class STUDENTS
*
IF NAME = 'John Smith'
  STREET := 'Oxford street'
  CITY   := 'London'
```

```
END-IF
END
```

プログラム **METH01** の出力：

```

Page      1                                                 05-01-17  15:59:04

Name  : John Smith
Street: Oxford street
City  : London

The summation of               1           2           3           4
is          10
The multiplication of          1           2           3           4
is          24
```

118 SEPARATE



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**COMPRESS** | **COMPUTE** | **EXAMINE** | **MOVE** | **MOVE ALL** | **RESET**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

SEPARATE ステートメントは１つの英数字またはバイナリオペランドの内容を複数の英数字またはバイナリオペランド（または英数字やバイナリ配列の複数オカレンス）に分割するために使用します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A	U				B							可	不可
operand2	C	S						N	P	I	B*							可	不可
operand3	C	S						N	P	I	B*							可	不可
operand4		S	A	G		A	U				B							可	可
operand5		S				A	U				B							可	可
operand6	C	S				A	U				B							可	不可
operand7		S						N	P	I								可	可

* *operand2* および *operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

operand1	ソースオペランド： *operand1* は、内容を分割する英数字／バイナリの定数または変数です。 *operand1* 内の末尾の空白は、その値が処理される前に削除されます（デリミタ文字として空白を使用する場合でも同様です。DELIMITERS オプションも参照）。
SUBSTRING	通常、フィールドの内容全体が先頭から分割されます。 SUBSTRING オプションでは、フィールドの一部だけを分割できます。 SUBSTRING 節のフィールド名（*operand1*）の後に、まず開始位置（*operand2*）、次に分割するフィールド部分の長さ（*operand3*）を指定します。 例えば、フィールド #A の値が "CONTRAPTION" の場合、SUBSTRING(#A,5,3) の内容は "RAP" を示します。 **注意:** *operand2* を省略すると、開始位置は "1" と見なされます。 *operand3* を省略すると、長さはフィールドの開始位置から終わりまでと見なされます。
LEFT JUSTIFIED	このオプションでは、ターゲットオペランドからデリミタ文字と次の空白以外の文字の間に存在する先頭の空白を削除します。
operand4	ターゲットオペランド： *operand4* は、ターゲットオペランドを表します。 ターゲットオペランドとして配列が指定されている場合、分割された値は各オカレンスに送られます。 ターゲットオペランドの個数は、*operand1* 内のデリミタ文字（末尾のデリミタ文字を含む）の個数に 1 を加えたものに相当します。 *operand4* がダイナミック変数の場合、SEPARATE 操作によってその長さを変更できます。 ダイナミック変数の現在の長さは、システム変数 *LENGTH を使用して確認できます。 ダイナミック変数に関する全般的な情報については、「*ラージ変数／フィールドとダイナミック変数／フィールド*」を参照してください。
IGNORE / REMAINDER *operand5*	分割されたソース値を受け取るターゲットフィールドが足りないと、該当するエラーメッセージが返されます。 これを避けるために、次の 2 つのオプションを使用できます。 ■ IGNORE を指定すると、Natural はソース値を受け取るターゲットオペランドが足りない場合はそれを無視します。 ■ REMAINDER *operand5* を指定すると、ターゲットオペランドに挿入できなかったソース値は *operand5* に挿入されます。 *operand5* の内容は、後続の SEPARATE ステートメントなどの処理に使用できます。 *例 3* も参照してください。
DELIMITERS	下記の「*デリミタオプション*」を参照してください。

WITH RETAINED DELIMITERS	通常、デリミタ文字自体はターゲットオペランドには移されません。 RETAINEDを指定すると、デリミタ（デフォルトのデリミタや空白、あるいは*operand6*で指定されたデリミタ）もターゲットオペランドに挿入されます。 例： 次の SEPARATE ステートメントでは、"150" を #B、"+" を #C、"30" を #D に挿入しています。 `...` `MOVE '150+30' TO #A` `SEPARATE #A INTO #B #C #D WITH RETAINED DELIMITER '+'` `...` *例 3* も参照してください。
GIVING NUMBER *operand7*	GIVING NUMBERオプションでは、値（空白で返されたものも含む）が挿入されたターゲットオペランドの個数を *operand7* に返します。 実際に得られる数はデリミタ文字の個数に 1 を加えたものです。 IGNORE オプションを使用した場合、*operand7* に返される有効な最大数はターゲットオペランド（*operand4*）の個数です。 REMAINDERオプションを使用した場合、*operand7*に返される有効な最大数はターゲットオペランド（*operand4*）の個数に *operand5* を加えたものです。

デリミタオプション：

```
[ WITH [RETAINED] { [ANY] DELIMITERS      } ]
                  { INPUT DELIMITERS      }
                  { DELIMITERS operand6   }
```

operand1 内のデリミタ文字は、値を分割する位置を示します。

- DELIMITERS オプションを省略した場合（または WITH ANY DELIMITERS を指定した場合）、空白、および英字や数字以外の任意の文字がデリミタ文字として扱われます。
- WITH INPUT DELIMITERS は、空白およびデフォルトの INPUT 区切り文字（セッションパラメータ ID で指定）がデリミタ文字として使用されることを示します。
- WITH DELIMITERS *operand6*は、指定した各文字（*operand6*）がデリミタ文字として使用されることを示します。 *operand6* に含まれる末尾の空白は無視されます。

例

例 1 - 各種サンプル

```
** Example 'SEPEX1': SEPARATE
************************************************************************
DEFINE DATA LOCAL
1 #TEXT1   (A6) INIT <'AAABBB'>
1 #TEXT2   (A7) INIT <'AAA BBB'>
1 #TEXT3   (A7) INIT <'AAA-BBB'>
1 #TEXT4   (A7) INIT <'A.B/C,D'>
1 #FIELD1A (A6)
1 #FIELD1B (A6)
1 #FIELD2A (A3)
1 #FIELD2B (A3)
1 #FIELD3A (A3)
1 #FIELD3B (A3)
1 #FIELD4A (A3)
1 #FIELD4B (A3)
1 #FIELD4C (A3)
1 #FIELD4D (A3)
1 #NBT     (N1)
1 #DEL     (A5)
END-DEFINE
*
WRITE NOTITLE 'EXAMPLE A (SOURCE HAS NO BLANKS)'
SEPARATE #TEXT1 INTO #FIELD1A #FIELD1B GIVING NUMBER #NBT
WRITE     / '=' #TEXT1 5X '=' #FIELD1A 4X '=' #FIELD1B 4X '=' #NBT
*
WRITE NOTITLE /// 'EXAMPLE B (SOURCE HAS EMBEDDED BLANK)'
SEPARATE #TEXT2 INTO #FIELD2A #FIELD2B GIVING NUMBER #NBT
WRITE     / '=' #TEXT2 4X '=' #FIELD2A 7X '=' #FIELD2B 7X '=' #NBT
*
WRITE NOTITLE /// 'EXAMPLE C (USING DELIMITER ''-'')'
SEPARATE #TEXT3 INTO #FIELD3A #FIELD3B WITH DELIMITER '-'
WRITE     /    '=' #TEXT3 4X '=' #FIELD3A 7X '=' #FIELD3B
*
MOVE ',/' TO #DEL
WRITE NOTITLE /// 'EXAMPLE D USING DELIMITER' '=' #DEL
*
SEPARATE #TEXT4 INTO #FIELD4A #FIELD4B
         #FIELD4C #FIELD4D WITH DELIMITER #DEL
WRITE     /    '=' #TEXT4 4X '=' #FIELD4A 7X '=' #FIELD4B
```

```
          /              19X '=' #FIELD4C 7X '=' #FIELD4D
*
END
```

プログラム **SEPEX1** の出力：

```
EXAMPLE A (SOURCE HAS NO BLANKS)

#TEXT1: AAABBB     #FIELD1A: AAABBB    #FIELD1B:           #NBT:  1



EXAMPLE B (SOURCE HAS EMBEDDED BLANK)

#TEXT2: AAA BBB    #FIELD2A: AAA       #FIELD2B: BBB       #NBT:  2



EXAMPLE C (USING DELIMITER '-')

#TEXT3: AAA-BBB    #FIELD3A: AAA       #FIELD3B: BBB



EXAMPLE D USING DELIMITER #DEL: ,/

#TEXT4: A.B/C,D    #FIELD4A: A.B       #FIELD4B: C
                   #FIELD4C: D         #FIELD4D:
```

例 2 - 配列の使用

```
** Example 'SEPEX2': SEPARATE (using array variable)
************************************************************************
DEFINE DATA LOCAL
1 #INPUT-LINE (A60) INIT <'VALUE1,   VALUE2,VALUE3'>
1 #FIELD      (A20/1:5)
1 #NUMBER     (N2)
END-DEFINE
*
SEPARATE #INPUT-LINE LEFT JUSTIFIED INTO #FIELD (1:5)
                     GIVING NUMBER IN #NUMBER
*
WRITE NOTITLE #INPUT-LINE //
              #FIELD (1)  /
              #FIELD (2)  /
              #FIELD (3)  /
              #FIELD (4)  /
              #FIELD (5)  /
              #NUMBER
```

```
*
END
```

プログラム **SEPEX2** の出力：

```
VALUE1,   VALUE2,VALUE3

VALUE1
VALUE2
VALUE3


  3
```

例 3 - REMAINDER/RETAINED オプションの使用

```
** Example 'SEPEX3': SEPARATE (with REMAINDER, RETAIN option)
************************************************************************
DEFINE DATA LOCAL
1 #INPUT-LINE (A60) INIT <'VAL1,   VAL2, VAL3,VAL4'>
1 #FIELD      (A10/1:4)
1 #REM        (A30)
END-DEFINE
*
WRITE TITLE LEFT 'INP:' #INPUT-LINE /
            '#FIELD (1)' 13T '#FIELD (2)' 25T '#FIELD (3)'
        37T '#FIELD (4)' 49T 'REMAINDER'
       /    '----------' 13T '----------' 25T '----------'
        37T '----------' 49T '------------------------------'
*
SEPARATE #INPUT-LINE INTO #FIELD (1:2)
         REMAINDER #REM WITH DELIMITERS ','
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
*
RESET #FIELD(*) #REM
SEPARATE #INPUT-LINE INTO #FIELD (1:2)
         IGNORE WITH DELIMITERS ','
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
*
RESET #FIELD(*) #REM
SEPARATE #INPUT-LINE INTO #FIELD (1:4) IGNORE
        WITH RETAINED DELIMITERS ','
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
*
RESET #FIELD(*) #REM
*
SEPARATE SUBSTRING(#INPUT-LINE,1,50) INTO #FIELD (1:4)
         IGNORE WITH DELIMITERS ','
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
```

```
*
END
```

プログラム **SEPEX3** の出力：

```
INP: VAL1,   VAL2, VAL3,VAL4
#FIELD (1)  #FIELD (2)  #FIELD (3)  #FIELD (4)  REMAINDER
----------  ----------  ----------  ----------  ------------------------------
VAL1           VAL2                              VAL3,VAL4
VAL1           VAL2
VAL1        ,              VAL2     ,
VAL1           VAL2       VAL3       VAL4
```

119 SET CONTROL

```
SET CONTROL operand1 ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

SET CONTROL ステートメントは、端末コマンドをプログラム内から実行するために使用します。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A												可	不可

構文要素の説明：

operand1	指定する端末コマンド： 端末コマンドは、制御文字 "%" なしで*operand1* として指定し（デフォルト）、テキスト定数または英数字変数の内容として指定できます。 端末コマンドの詳細については、『*端末コマンド*』ドキュメントを参照してください。

例

- 例 1 - 小文字への変換

例 1 - 小文字への変換

```
...
SET CONTROL 'L'
...
```

小文字に変換します（端末コマンド `%L` に相当）。

例 2 - ハードコピー出力先の有効化

```
...
SET CONTROL 'HDEST'
...
```

ハードコピーの出力先を "DEST" にします（端末コマンド `%H`*`destination`* に相当）。

120 SET GLOBALS

```
SET GLOBALS parameter=value ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

SET GLOBALS ステートメントは、セッションパラメータの値を設定するために使用します。

パラメータは、SET GLOBALS ステートメントを含むプログラムのコンパイル時または実行時に評価されます。どちらで評価されるかは、個々のパラメータよって異なります。

SET GLOBALS で指定したパラメータ設定は、後続の SET GLOBALS ステートメント（または GLOBALS システムコマンド）で上書きされない限り、Natural セッションが終了するまで有効です。 SET GLOBALS ステートメントおよび GLOBALS システムコマンドには、同じ修正用パラメータが用意されています。 両方とも同じ Natural セッション内で使用できます。 GLOBALS コマンドで指定したパラメータ値は、新しい GLOBALS コマンドまたは SET GLOBALS ステートメントで上書きされるか、セッションが終了するか、またはユーザーが別のライブラリにログオンするまで有効です。

パラメータ

複数のパラメータを指定する場合、1つ以上の空白で区切る必要があります。パラメータは任意の順序で指定できます。*例*を参照してください。

SET GLOBALS ステートメントで設定可能なパラメータ	評価（R＝ランタイム、C＝コンパイル時）
CF - 端末コマンドの文字	R
CPCVERR - コードページ変換エラー	R
DC - 小数点表記の文字	R
DC - 小数点表記の文字	R
DFOUT - 出力の日付フォーマット	R
DFSTACK - スタックの日付フォーマット	R
DFTITLE - デフォルトページタイトルの日付フォーマット	R
DU - ダンプ生成	R
EJ - ページ換え	R
FCDP - ダイナミックに保護されたフィールドの充填文字	R
FS - フォーマット指定	R
IA - INPUT 割り当て文字	R

SET GLOBALS ステートメントで設定可能なパラメータ	評価（R=ランタイム、C=コンパイル時）
ID - INPUT 区切り文字	R
IM - 入力モード	R
LE - エラー処理の制限	C
LS - 行サイズ	C
LT - レコードの読み取り制限	R
NC - Natural システムコマンドの使用	R
OPF - ヘルプルーチンによる保護されたフィールドの上書き	R
PM - 出力モード	C
PS - ページサイズ	RC
REINP - 不正データに対する内部的な REINPUT	R
SA - サウンド端末アラーム	R
SF - フィールド間の空白	C
WH - ホールド状態でのレコードの待機	R
ZD - ゼロ割り算のチェック	R
ZP - ゼロ出力	R

各セッションパラメータの詳細については、『パラメータリファレンス』を参照してください。

例

次の例では、`SET GLOBALS` ステートメントを使用して、1 行当たりの最大文字数を 74 文字に設定し、Natural プログラム内の処理ループで読み取ることができるデータベースレコードの数を 5000 に制限しています。

```
SET GLOBALS LS=74 LT=5000
...
```

121 SET KEY

このchapterでは、次のトピックについて説明します。

機能

SET KEY ステートメントは、次のタイプのキーに機能を割り当てるために使用します。

- ビデオ端末 PA（プログラムアテンション）キー
- PF（プログラムファンクション）キー
- CLEAR キー

SET KEY ステートメントを実行すると、Natural はプログラム実行中にキーの制御を受け取り、各キーに割り当てられた値を使用します。

Natural システム変数 *PF-KEY は、最後に押されたキーを示します。

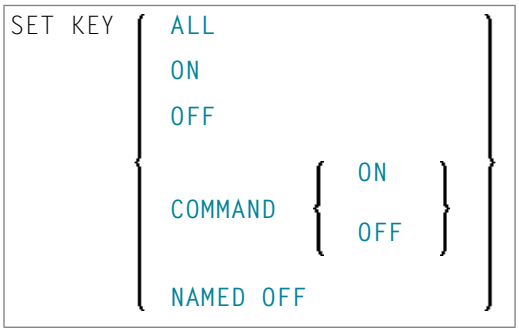
Note: 機能が割り当てられていないキーを押すと、有効なキーを押すように促す警告メッセージが発行されるか、または Natural システム変数 *PF-KEY に値 "ENTR" が設定されます。つまり、Natural は Enter キーが押されたときと同様に処理します。これは、Natural 管理者が Natural プロファイルパラメータ IKEY をどのように設定したかによります。

構文説明

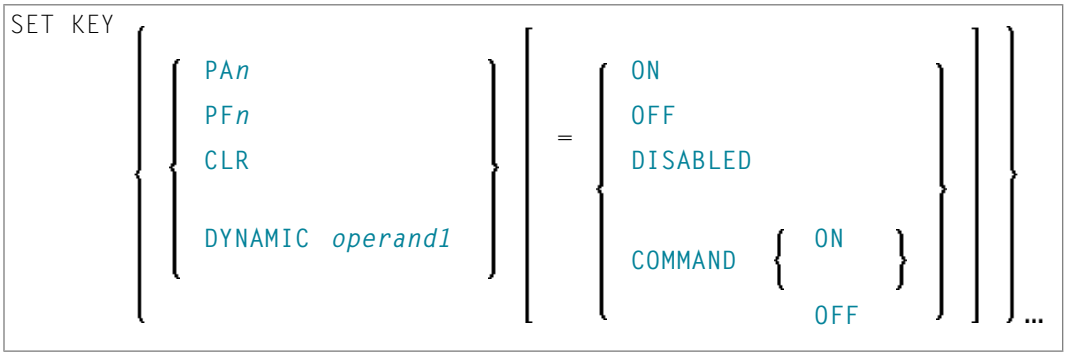
このステートメントには、いくつかの構造を使用できます。

構文図で使用されている記号については、「*構文記号*」を参照してください。

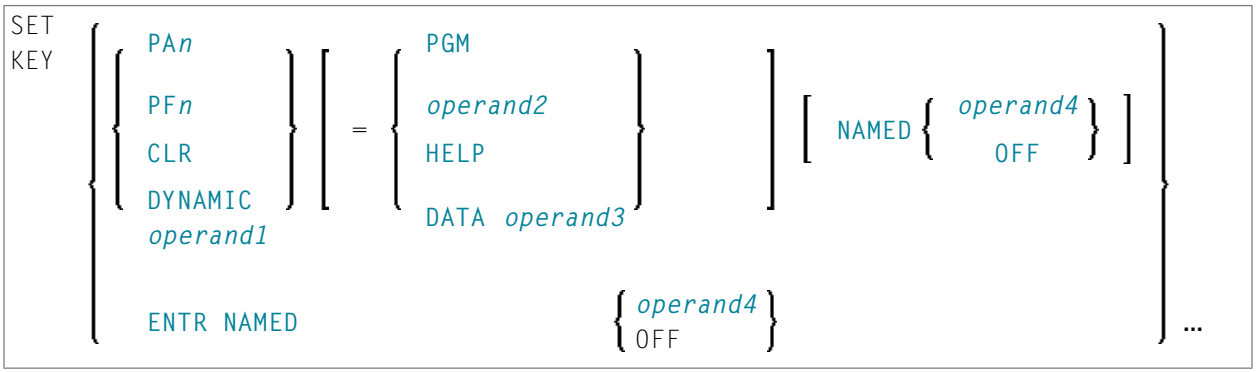
構文 1 - すべてのキーに有効：



構文 2 - 個々のキーに有効：



構文 3 - 個々のキーに有効：



オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1		S				A												可	不可
operand2	C	S				A	U											可	不可
operand3	C	S				A	U											可	不可
operand4	C	S				A	U											可	不可

キーのプログラム察知の有効化／キーの解除

キーをプログラム察知可能にするとは、現在アクティブなプログラムによるキーの問い合わせを可能にするということです。 キーをプログラム察知可能にすると、キーの押下は Enter キーを押したときと同様の効果を持ちます。 画面に入力されたすべてのデータがプログラムに転送されます。

Note: PA キーや CLEAR キーをプログラム察知可能にしたときは、画面のデータは転送されません。

プログラム察知は、現在のプログラムの実行に対してのみ有効です。 「*異なるプログラムレベルでの SET KEY ステートメント*」も参照してください。

例：

SET KEY ALL	このステートメントでは、すべてのキーをプログラム察知可能にします。 各キーに割り当てられたすべての機能が上書きされます。
SET KEY PF2 **SET KEY PF2=PGM**	これらの各ステートメントでは、PF2 をプログラム察知可能にします。
SET KEY OFF	このステートメントでは、すべてのキー設定を解除します。 SET KEY OFF の実行後、システム変数 *PF-KEY に "ENTR" が設定されます。
SET KEY ON	このステートメントでは、設定が解除される前にすべてのキーに割り当てられていた機能を再び有効にし、プログラム察知可能だったすべてのキーを察知可能にします。
SET KEY PF2=OFF	このステートメントでは、PF2 の設定を解除します。 SET KEY PF2=OFF の実行後、Natural システム変数 *PF-KEY に "ENTR" が設定されます（今まで "PF2" が設定されていた場合）。
SET KEY PF2=ON	このステートメントでは、設定が解除される前に PF2 に割り当てられていた機能を再び有効にし、プログラム察知可能にします。 PF2 に機能が割り当てられていなかった場合は、再びプログラム察知可能にします。

キーのプログラム察知と *PF-KEY の内容

次の例は、キーのプログラム察知とシステム変数 *PF-KEY の内容の関係を示しています。

PF2 が SET KEY PF2=PGM によりプログラム察知可能になっていて、後で INPUT ステートメントが実行されることを前提としています。 次の表は、ユーザーアクションと実行される Natural ステートメントが *PF-KEY の内容にどのように影響するかを示しています。

シーケンス	実行される Natural ステートメント／ユーザーアクション	*PF-KEY の内容
1	ユーザーによる PF2 の押下	"PF2"
2	SET KEY OFF	"ENTR"
3	SET KEY ON	"PF2"
4	SET KEY PF2=OFF	"ENTR"
5	SET KEY PF2=ON	"PF2"
6	SET KEY PF3=OFF	"PF2"

コマンド／プログラムの割り当て

キーにコマンドまたはプログラム名を割り当てることができます。 キーが押されると、現在のプログラムは終了し、そのキーに割り当てられているコマンド／プログラムがNaturalスタックを介して呼び出されます。 コマンド／プログラムを割り当てる場合、コマンド／プログラムにパラメータを渡すこともできます（下記の 3 番目の例を参照）。

端末コマンドをキーに割り当てることもできます。 キーが押されると、キーに割り当てられている端末コマンドが実行されます。

operand2 を定数として指定する場合は、一重引用符で囲む必要があります。

例：

`SET KEY PF4='SAVE'`	コマンド `SAVE` が PF4 に割り当てられます。
`SET KEY PF4=#XYX`	変数 `#XYZ` の値が PF4 に割り当てられます。
`SET KEY PF6='LIST MAP *'`	コマンド `LIST`（`LIST` パラメータ `MAP` および `*` を含む）が PF6 に割り当てられます。
`SET KEY PF2='%%'`	端末コマンド `%%` が PF2 に割り当てられます。

この割り当ては、別の `SET KEY` ステートメントで上書きされるまで、別のアプリケーションにログオンするまで、または Natural セッションが終了するまで有効です。 「*異なるプログラムレベルでの **SET KEY** ステートメント*」も参照してください。

Note: キーを通して呼び出されたプログラムを実行する前に、Natural は `BACKOUT TRANSACTION` ステートメントを内部的に発行します。

入力データの割り当て

データ文字列（*operand3*）をキーに割り当てることができます。 キーが押されると、カーソルが現在置かれている入力フィールドにデータ文字列が挿入され、データは実行中のプログラムに転送されます（Enter キーが押された場合と同様）。

operand3 を定数として指定する場合は、一重引用符で囲む必要があります。

例：

```
SET KEY PF12=DATA 'YES'
```

DATA 割り当ての整合性については、コマンド割り当てと同様のことが適用されます。つまり、別の SET KEY ステートメントで上書きされるまで、別のアプリケーションにログオンするまで、または Natural セッションが終了するまで有効です。 「*異なるプログラムレベルでの SET KEY ステートメント*」も参照してください。

COMMAND OFF/ON

COMMAND OFF では、キーに割り当てられている各機能（コマンド、プログラム、データ）を一時的に解除できます。機能が割り当てられる前にキーがプログラム察知可能であった場合、COMMAND OFF によってそのキーが再び察知可能になります。

その後に COMMAND ON を使用して、割り当てられていた機能を再び有効にすることができます。

例：

SET KEY PF4=COMMAND OFF	PF4 に割り当てられている機能が一時的に解除されます。機能が割り当てられる前に PF4 がプログラム察知可能であった場合、再び察知可能になります。
SET KEY PF4=COMMAND ON	PF4 に割り当てられていた機能が再び有効になります。
SET KEY COMMAND OFF	すべてのキーに割り当てられている全機能が一時的に解除されます。機能が割り当てられる前にこれらのキーがプログラム察知可能であった場合、再び察知可能になります。
SET KEY COMMAND ON	すべてのキーに割り当てられていた全機能が再び有効になります。

SET KEY PF*nn*='' では、キーに割り当てられている機能を削除し、それと同時にキーのプログラム察知を解除できます。

HELP の割り当て

"HELP" をキーに割り当てることができます。キーが押されると、カーソルが現在置かれているフィールドに割り当てられているヘルプルーチンが呼び出されます。

これはヘルプを呼び出すフィールドにヘルプ文字を入力した場合と同様です。 ヘルプ文字（デフォルトは "?"）は、Natural プロファイルパラメータ HI（Natural 管理が設定）によって決定されます。

例：

```
SET KEY PF1=HELP
```

HELP 割り当ての整合性については、プログラム察知と同様のことが適用されます。つまり、現在のプログラムの実行にのみ有効です。 「*異なるプログラムレベルでの SET KEY ステートメント*」も参照してください。

DYNAMIC オプション

SET KEY ステートメントで特定のキーを指定する代わりに、DYNAMIC オプションを変数（ *operand1* ）とともに使用し、プログラム内でこの変数に値（PF*n*、PA*n*、CLR）を割り当てることができます。 これにより、どのキーに機能を割り当てるかをプログラムロジックによって指定できます。

例：

```
...
IF ...
   MOVE 'PF4' TO #KEY
ELSE
   MOVE 'PF7' TO #KEY
END-IF
...
SET KEY DYNAMIC #KEY = 'SAVE'
...
```

DISABLED オプション

プッシュボタン、メニュー、ビットマップなどのグラフィカルユーザーインターフェイス（GUI）エレメントは PF キーとして実装されています。 DISABLED オプションでは、PF キーに割り当てられた GUI エレメントを使用できないようにします。 これにより、プッシュボタンとメニュー項目が灰色表示されます。

SET KEY PF*nn*=DISABLED の設定を解除するには、SET KEY PF*nn*=ON を使用します。

例：

`SET KEY PF10=DISABLED`	PF10 に割り当てられた GUI エレメントを使用できないようにします。

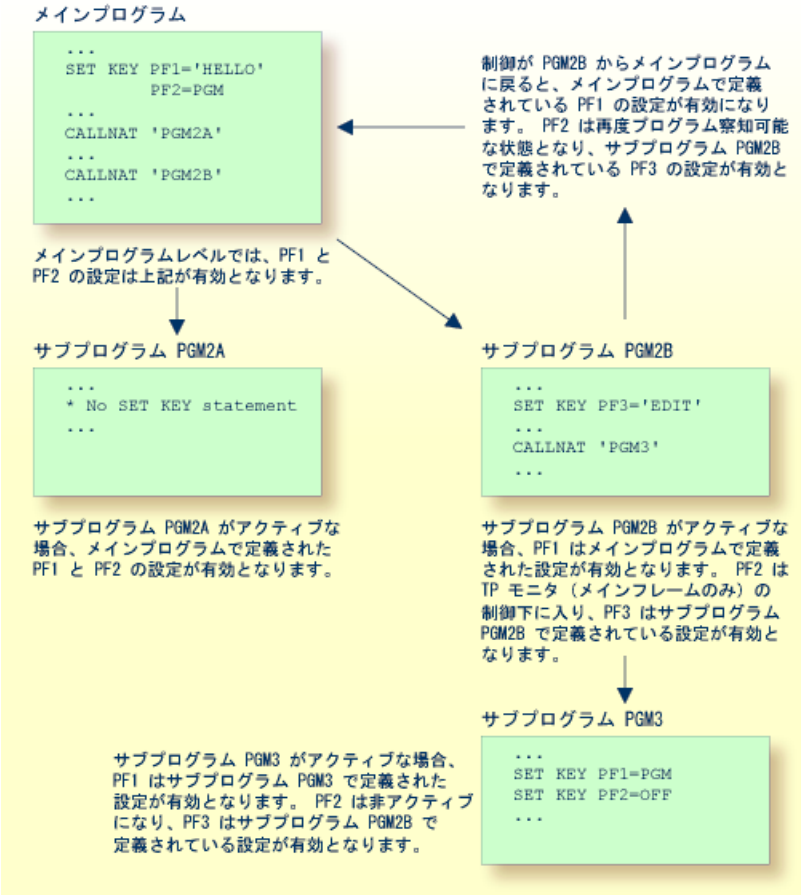
`DISABLED` オプションは処理ルール内でのみ使用できます。

異なるプログラムレベルでの SET KEY ステートメント

アプリケーション内の異なるレベルで `SET KEY` ステートメントが指定されている場合、次のことが適用されます。

- キーがプログラム察知可能な場合、各プログラムで別の `SET KEY` ステートメントが指定されるまで、下位レベルの（呼び出される）全プログラムにもこのプログラム察知が適用されます。 制御が上位レベルのプログラムに返されると、上位レベルでの `SET KEY` 割り当てが再び有効になります。
- `HELP` キーとして定義されたキーについては、プログラム察知可能なキーと同様のことが適用されます。
- キーに機能（プログラム、コマンド、端末コマンド、データ文字列）が割り当てられると、この割り当ては（割り当てが行われたレベルに関係なく）すべての上位レベルおよび下位レベルで有効です。これは、そのキーに別の機能が割り当てられるか、キーがプログラム察知可能になるまで、別のアプリケーションにログオンするまで、またはNaturalセッションが終了するまで有効です。

異なるプログラムレベルでの SET KEY ステートメントの例



名前の割り当て

NAMED 節を使用して、キーに名前（ *operand4* ）を割り当てることができます。 名前は画面上の PF キー行に表示されます。これにより、キーに割り当てられている機能を識別できます。

```
                ?   Help
              .   Exit
             ----  ------------------------------------------------------
     Code ..: ?    Library ..: *_______
                   Object ...: *_____________________________________________
                   DBID .....: 0__     FILENR ...: 0__


  Command ===>
```

```
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       Help        Exit  Last        Flip                                  Canc
```

PF キー行の表示はセッションパラメータ KD（『パラメータリファレンス』ドキュメントを参照）で有効になります。

キーに割り当てる名前の最大長は 10 文字です。 標準タブ PF キー行形式では、最初の 5 文字だけが表示されます。

*operand4*を定数として指定する場合は、一重引用符で囲む必要があります（下記の例を参照）。

機能を割り当てていないキーやプログラム察知可能でないキーに名前を割り当てることはできません。 ただし、Enter キーだけには機能を割り当てなくても名前を割り当てることができます。

NAMED OFF では、プログラム察知可能なキーに割り当てた名前を削除できます。

例：

`SET KEY ENTR NAMED 'EXEC'`	Enter キーに "EXEC" という名前を割り当てます。
`SET KEY PF3 NAMED 'EXIT'`	PF3 をプログラム察知可能にし、PF3 に "EXIT" という名前を割り当てます。
`SET KEY PF3 NAMED OFF`	PF3 をプログラム察知可能にし、PF3 に割り当てられている名前を削除します。
`SET KEY NAMED OFF`	プログラム察知可能な各キーに割り当てられているすべての名前を削除します。
`SET KEY PF4='AP1' NAMED 'APPL1'`	PF4 にプログラム "AP1" および名前 "APPL1" を割り当てます。

標準タブ PF キー行形式 を使用する場合、次のことが適用されます。

- キーにコマンド／プログラムを割り当てる際に NAMED 節を省略すると、コマンド／プログラム名が PF キー行に表示されます。コマンド／プログラム名が 5 文字を超過すると、"CMND" が表示されます。
- キーに入力データを割り当てる際に NAMED 節を省略すると、"DATA" が PF キー行に表示されます。
- （NAMED 節を使用して）名前を Unicode フォーマットで PF キーに割り当てる場合、名前が対応するヘッダーに正しく配置されないことがあります。ただし、この問題は、*Web I/O インターフェイス*を使用している場合、およびワイド文字に対してのみ発生します。この場合、順次 PF キー行形式を使用することをお勧めします。

順次 PF キー行形式を使用すると、名前を割り当てたキーのみが PF キー行に表示されます。つまり、キーにコマンド／プログラム／データを割り当てる際に NAMED 節を省略すると、キーは PF キー行に表示されません。

例

```
** Example 'SKYEX1': SET KEY
************************************************************************
DEFINE DATA LOCAL
1 #PF4 (A56)
END-DEFINE
*
MOVE 'LIST VIEW' TO #PF4
*
SET KEY PF1 PF2
SET KEY PF3 = 'MENU'
        PF4 = #PF4
        PF5 = 'LIST VIEW EMPLOYEES' NAMED 'Empl'
*
FORMAT KD=ON
INPUT ////
      10X 'The following function keys are assigned:' //
      10X 'PF1: Function for PF1      ' /
      10X 'PF2: Function for PF2      ' /
      10X 'PF3: Return to MENU program' /
      10X 'PF4: LIST VIEW             ' /
      10X 'PF5: LIST VIEW EMPLOYEES   ' ///
*
IF *PF-KEY = 'PF1'
  WRITE 'Function for PF1 executed.'
END-IF
IF *PF-KEY = 'PF2'
  WRITE 'Function for PF2 executed.'
END-IF
*
END
```

プログラム **SKYEX1** の出力：

```
The following function keys are assigned:

PF1: Function for PF1
PF2: Function for PF2
PF3: Return to MENU program
PF4: LIST VIEW
PF5: LIST VIEW EMPLOYEES
```

122 SET TIME

```
{ SET TIME }
{ SETTIME  }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

SET TIME（またはSETTIME）ステートメントは、Naturalシステム変数*TIMDと組み合わせて、プログラムの特定のセクションを実行するために要した時間を計測するために使用します。

SET TIMEステートメントはプログラムの特定の位置に指定されます。*TIMDの値は、SET TIMEステートメントの実行時点からの経過時間です。

*TIMDには必ずSET TIMEステートメントへの参照を含める必要があります。そのためには、SET TIMEステートメントのソースコード行番号を使用するか、参照として使用できるラベルをSET TIMEステートメントに割り当てます。

例

```
** Example 'STIEX1': SETTIME
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
ST. SETTIME
WRITE 10X 'START TIME:' *TIME
*
READ (100) EMPLOY-VIEW BY NAME
END-READ
*
WRITE NOTITLE 10X 'END TIME:  ' *TIME
WRITE         10X 'ELAPSED TIME TO READ 100 RECORDS'
                  '(HH:II:SS.T) :' *TIMD (ST.) (EM=99:99:99'.'9)
*
END
```

プログラム **STIEX1** の出力：

```
START TIME: 16:39:07.6
END TIME:   16:39:07.7
ELAPSED TIME TO READ 100 RECORDS (HH:MM:SS.T) : 00:00:00.1
```

123 SET WINDOW

```
SET WINDOW  { 'window-name' }
            {     OFF       }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DEFINE WINDOW** | **INPUT WINDOW='*window-name*'** | **REINPUT** | **SET WINDOW**

関連機能グループ：「*対話型処理用の画面生成*」

機能

SET WINDOW ステートメントは、ウィンドウをアクティブにしたり非アクティブにするために使用します。

SET WINDOW '*window-name*' または INPUT WINDOW='*window-name*' ステートメントでは、現在アクティブなウィンドウを非アクティブにし、ステートメントで指定したウィンドウをアクティブにします。 一度にアクティブにできるウィンドウは 1 つだけです。

Note: SIZE AUTO で定義されたウィンドウを SET WINDOW でアクティブにすると、ウィンドウがアクティブになる*前*の画面上のデータによってウィンドウのサイズが決定されます。

構文説明

SET WINDOW '*window-name*'	指定したウィンドウをアクティブにします。つまり、そのウィンドウを非アクティブにするか、別のウィンドウをアクティブにするまで、後続のステートメントはすべてそのウィンドウを参照します。 指定するウィンドウは DEFINE WINDOW ステートメントで定義されている必要があります。
SET WINDOW OFF	現在アクティブなウィンドウを非アクティブにします。

例

`DEFINE WINDOW` ステートメントを参照してください。

124 SKIP

```
SKIP [(rep)] operand1 [LINES]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「構文記号」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

SKIP ステートメントは、1 つまたは複数の空行を出力レポートに生成するために使用します。

『*ページタイトル、改ページ、空行*』（『*プログラミングガイド*』）も参照してください。

処理

SKIP ステートメントの実行によってページサイズを超過してしまった場合、超過分の行は無視されます（AT TOP OF PAGE ステートメントの場合を除く）。

SKIP ステートメントは、ページにすでに何らかの出力がある場合にのみ実行されます（AT TOP OF PAGE ステートメントからの出力を除く）。

構文説明

オペランド定義テーブル：

オペランド	構文要素						フォーマット									ステートメント参照	ダイナミック定義
operand1	C	S					N	P	I							可	不可

構文要素の説明：

(*rep*)	レポート指定： 表記 (*rep*) を使用して、SKIP ステートメントを適用するレポートの ID を指定できます。 範囲 0～31 の値、または **DEFINE PRINTER** ステートメントを使用して割り当てた論理名を指定できます。 (*rep*) 指定がない場合、SKIP ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。
operand1	省略される行数： *operand1* は、生成する空行の数（1～250）を表します。 この数は、数値定数または数値変数の内容として指定できます。 *operand1* がレポートのページサイズを超過した場合、SKIP ステートメントは NEWPAGE 条件と同じ結果になります。

例

```
** Example 'SKPEX1': SKIP
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
  2 NAME
END-DEFINE
*
LIMIT 7
READ EMPL-VIEW BY CITY STARTING FROM 'W'
  AT BREAK OF CITY
    SKIP 2
  END-BREAK
  DISPLAY NOTITLE CITY (IS=ON) COUNTRY (IS=ON) NAME
  /*
END-READ
END
```

プログラム **SKPEX1** の出力：

```
        CITY          COUNTRY         NAME
-------------------- ------- --------------------

WASHINGTON           USA     REINSTEDT
                             PERRY


WEITERSTADT          D       BUNGERT
                             UNGER
                             DECKER


WEST BRIDGFORD       UK      ENTWHISTLE


WEST MIFFLIN         USA     WATSON
```

125 SORT

ストラクチャードモード構文

```
END-ALL
[AND]
SORT      [ THEM    ] [BY] { operand1 [ ASCENDING  ] }... 10
          [ RECORDS ]                  [ DESCENDING ]
            USING-clause
            [GIVE-clause]
            statement ...
END-SORT
```

* ステートメントラベルを指定する場合、キーワード SORT の*前*、かつ END-ALL（および AND）の*後*に指定する必要があります。

レポーティングモード構文

```
SORT [ THEM    ] [BY] { operand1 [ ASCENDING  ] }... 10
     [ RECORDS ]                 [ DESCENDING ]
       USING-clause
       [GIVE-clause]
       statement ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：***FIND（SORTED BY オプション付き）***

機能

SORT ステートメントは、ソート処理を実行するために使用します。SORT ステートメントを実行すると、アクティブなすべての処理ループからのレコードがソートされます。

ソート処理には、Naturalの内部ソートプログラムが使用されます。別の外部ソートプログラムを使用することもできます。

制限事項

- SORT ステートメントは、ソートするレコードを持つ処理ループと同じオブジェクト内に指定する必要があります。
- SORT ステートメントのネスト構造は許可されていません。
- ソートするレコードの合計長が 10240 バイトを超過しないようにする必要があります。
- ソート条件の数は 10 を超過しないようにする必要があります。

構文説明

オペランド定義テーブル：

オペランド	構文要素	フォーマット	ステートメント参照	ダイナミック定義
operand1	S	A N P I F B D T	不可	不可

構文要素の説明：

END-ALL	ストラクチャードモードでは、SORT ステートメントの前に必ず END-ALL ステートメントを指定して、アクティブなすべての処理ループを終了する必要があります。 SORT ステートメント自体では新しい処理ループを開始し、END-SORT ステートメントでループを終了する必要があります。 **注意:** レポーティングモードでは、SORT ステートメントはアクティブなすべての処理ループを終了させ、新しい処理ループを開始します。
operand1	ソート条件： *operand1* は、ソート条件として使用するフィールド／変数を表します。 1～10 個のデータベースフィールド（ディスクリプタおよび非ディスクリプタ）またはユーザー定義変数を指定できます。マルチプルバリューフィールドまたはピリオディックグループ内のフィールドを使用できます。 グループまたは配列は指定できません。
ASCENDING **DESCENDING**	デフォルトのソート順は昇順です。 値を降順でソートする場合、DESCENDING を指定します。 ASCENDING/DESCENDING は、各ソートフィールドに指定できます。
USING	下記の「***USING 節***」を参照してください。
GIVE	下記の「***GIVE 節***」を参照してください。
END-SORT	SORT ステートメントを終了するには、Natural 予約語 END-SORT を使用する必要があります。

USING 節

USING節では、中間ソートストレージに書き込むフィールドを指定します。レポーティングモードでは任意で、ストラクチャードモードでは必須です。 ただし、これにより必要なメモリが軽減されるため、レポーティングモードでも使用することを強くお勧めします。

```
{ USING {operand2}... }
{ USING KEYS           }
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット											ステートメント参照	ダイナミック定義
operand2		S	A			A	N	P	I	F	B	D	T	L	C			不可	不可

構文要素の説明：

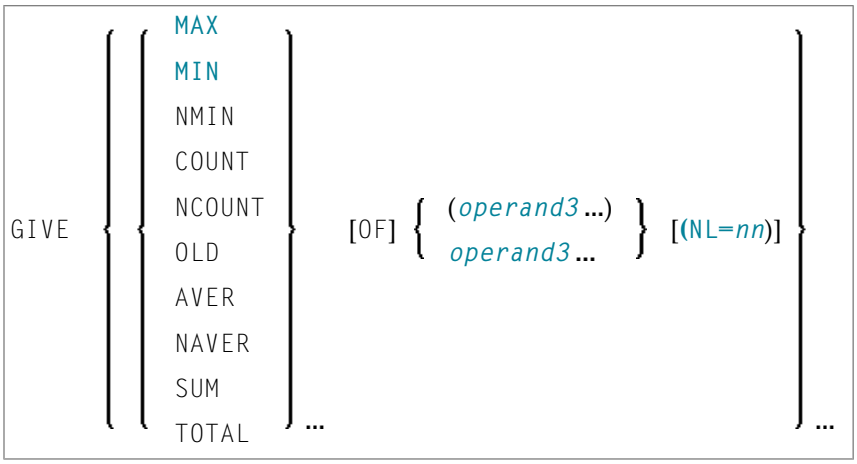
USING *operand2*	ソートキーフィールド（*operand1*で指定）に加えて、中間ソートストレージに書き込む追加のフィールドを指定できます。
USING KEYS	*operand1*で指定したソートキーフィールドのみが中間ソートストレージに書き込まれます。

*レポーティングモードの場合：*USING 節を省略した場合、SORT ステートメントの前に開始された処理ループのすべてのデータベースフィールドおよび SORT ステートメントの前に定義されたすべてのユーザー定義変数が中間ソートストレージに書き込まれます。

ソートの実行終了後、中間ソートストレージに書き込まれなかったフィールドを参照した場合、フィールドの値はソート前のフィールドの最終値になります。

GIVE 節

GIVE 節は、SORT ステートメントの第 1 段階で評価される Natural システム関数（MAX、MIN など）を指定するために使用します。これらのシステム関数を第3段階で参照できます（「*SORT ステートメント処理*」を参照）。 SORT ステートメントの後のシステム関数を参照するときには、*AVER(SALARY) のように、前にアスタリスクを付ける必要があります。



オペランド定義テーブル：

オペランド	構文要素	フォーマット	ステートメント参照	ダイナミック定義
operand3	S A	*	可	不可

* 使用するシステム関数に依存

構文要素の説明：

MAX | MIN | NMIN | COUNT | NCOUNT | OLD | AVER | NAVER | SUM | TOTAL	各システム関数の詳細については、『*システム関数*』ドキュメントを参照してください。
operand3	*operand3* はフィールド名です。
(NL=*nn*)	このオプションはAVER、NAVER、SUM、およびTOTALにのみ適用され、他のシステム関数では無視されます。 このオプションを使用して、システム関数の評価中の桁あふれを防ぐことができます。『*システム関数*』ドキュメントの「*AVER、NAVER、SUM、TOTALでの桁あふれ*」を参照してください。

SORT ステートメント処理（3 段階）

SORT ステートメントを含むプログラムは、3 つの段階で実行されます。

第 1 段階 - ソートするレコードの選択

SORT ステートメントより前のステートメントが実行されます。 USING 節に指定されたデータが中間ソートストレージに書き込まれます。

レポーティングモードでは、ソートの後で累計用に使用する変数は、SORT ステートメントの前で定義しないでください。 ストラクチャードモードでは、それらを USING 節に指定しないでください。 中間ソートストレージに書き込まれるフィールドを累計用として使用することはできません。処理の第 3 段階で個々のレコードが読み直されるためです。 各レコードについて、フィールドが読み直された個々のレコードの値で上書きされるため、累計機能として正しく処理されません。

中間ストレージに書き込まれるレコードの数は、処理ループの回数とループ当たりの処理レコードの数によって決まります。 処理ループで SORT ステートメントが出現するたびに内部中間ストレージに 1 レコードが作成されます。 ネスト構造のループの場合、内側のループが実行される場合にのみ、レコードは中間ストレージに書き込まれます。 次の例で、内側の（FIND）ループで該当レコードが見つからない場合でも中間ストレージにレコードを書き込むためには、FIND ステートメントに IF NO RECORDS FOUND 節を指定しないでください。

```
READ ...
  ...
  FIND ...
...
END-ALL
SORT ...
  DISPLAY ...
END-SORT
...
```

第 **2** 段階 **-** レコードのソート

レコードがソートされます。

第 **3** 段階 **-** ソートされたレコードの処理

SORT ステートメントより後のステートメントが、指定のソート順で中間ストレージのすべてのレコードに対して実行されます。 SORT ステートメントの後でデータベースフィールドを参照するには、適切なステートメントラベルまたは参照番号を使用して正しく参照する必要があります。

例

- 例 1 - SORT

例 1 - SORT

```
** Example 'SRTEX1S': SORT (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 SALARY      (1:2)
  2 PERSONNEL-ID
  2 CURR-CODE   (1:2)
*
1 #AVG          (P11)
1 #TOTAL-TOTAL  (P11)
1 #TOTAL-SALARY (P11)
1 #AVER-PERCENT (N3.2)
END-DEFINE
*
LIMIT 3
FIND EMPL-VIEW WITH CITY = 'BOSTON'
  COMPUTE #TOTAL-SALARY = SALARY (1) + SALARY (2)
  ACCEPT IF #TOTAL-SALARY GT 0
  /*
END-ALL
AND
SORT BY PERSONNEL-ID USING #TOTAL-SALARY SALARY(*) CURR-CODE(1)
     GIVE AVER(#TOTAL-SALARY)
  /*
  AT START OF DATA
    WRITE NOTITLE '*' (40)
         'AVG CUMULATIVE SALARY:' *AVER (#TOTAL-SALARY) /
    MOVE *AVER (#TOTAL-SALARY) TO #AVG
  END-START
  COMPUTE ROUNDED #AVER-PERCENT = #TOTAL-SALARY / #AVG * 100
  ADD #TOTAL-SALARY TO #TOTAL-TOTAL
  /*
  DISPLAY NOTITLE PERSONNEL-ID SALARY (1) SALARY (2)
          #TOTAL-SALARY CURR-CODE (1)
          'PERCENT/OF/AVER' #AVER-PERCENT
  AT END OF DATA
    WRITE / '*' (40) 'TOTAL SALARIES PAID: ' #TOTAL-TOTAL
  END-ENDDATA
END-SORT
*
END
```

プログラム **SRTEX1S** の出力：

```
PERSONNEL   ANNUAL     ANNUAL   #TOTAL-SALARY CURRENCY PERCENT
     ID       SALARY     SALARY                    CODE     OF
                                                          AVER
  --------- ---------- ---------- ------------- -------- -------

  ************************************** AVG CUMULATIVE SALARY:        41900

  20007000       16000      15200        31200  USD        74.00
  20019200       18000      17100        35100  USD        83.00
  20020000       30500      28900        59400  USD       141.00

  ************************************** TOTAL SALARIES PAID:        125700
```

上記の例は次のように実行されます。

第 **1** 段階：

- EMPLOYEES ファイルから CITY=BOSTON のレコードが選択されます。
- SALARY の最初の 2 オカレンスがフィールド #TOTAL-SALARY に加算されます。
- #TOTAL-SALARY で、0 より大きいレコードのみが受け入れられます。
- レコードが中間ソートストレージに書き込まれます。 データベース配列 SALARY（最初の 2 オカレンス）、CURR-CODE（第 1 オカレンス）、データベースフィールド PERSONNEL-ID、およびユーザー定義変数 #TOTAL-SALARY が中間ストレージに書き込まれます。
- #TOTAL-SALARY の平均値が評価されます。

第 **2** 段階：

- レコードがソートされます。

第 **3** 段階：

- ソートされた中間ストレージが読み取られます。
- AT START OF DATA 条件で #TOTAL-SALARY の平均値が表示されます。
- #TOTAL-SALARY が #TOTAL-TOTAL に加算され、フィールド PERSONNEL-ID、SALARY(1)、SALARY(2)、#AVER-PERCENT、および #TOTAL-SALARY が表示されます。
- AT END OF DATA 条件で変数 #TOTAL-TOTAL が出力されます。

レポーティングモードの例はライブラリ SYSEXRM のプログラム **SRTEX1R** を参照してください。

例 2 - SORT

```
** Example 'SRTEX2': SORT
************************************************************************
DEFINE DATA LOCAL
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
  2 YEAR
END-DEFINE
*
LIMIT 10
*
READ VEHIC-VIEW
END-ALL
SORT BY MAKE YEAR USING KEY
  DISPLAY NOTITLE (AL=15) MAKE (IS=ON) YEAR
  AT BREAK OF MAKE
    WRITE '-' (20)
  END-BREAK
END-SORT
END
```

プログラム **SRTEX2S** の出力：

```
     MAKE       YEAR
--------------- -----

FIAT             1980
                 1982
                 1984
--------------------
PEUGEOT          1980
                 1982
                 1985
--------------------
RENAULT          1980
                 1980
                 1982
```

```
                  1982
--------------------
```

外部ソートプログラムの使用

Natural では、ソート処理はデフォルトで Natural の内部ソートプログラムによって実行されます（上記参照）。 ただし、外部ソートプログラムを使用することもできます。 この外部ソートプログラムでは、Natural の内部ソートプログラムの代わりにソート処理を実行します。

外部ソートプログラムとして、SyncSort for Windows がサポートされています。 Windows に SyncSort for Windows がインストールされている場合、Natural によって外部ソートプログラムが呼び出され、ソート処理用に使用されます。

126 STACK

```
STACK [TOP]  { COMMAND operand1 [operand2 [(parameter)]] ...
             { [DATA] [FORMATTED] {operand2 [(parameter)]} ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**INPUT** | **RELEASE**

機能

STACK ステートメントは、Natural スタックに次のいずれかを挿入するために使用します。

- 実行する Natural プログラムまたは Natural システムコマンドの名前
- INPUT ステートメントの実行中に使用されるデータ

スタックの詳細については、「*プログラミングのその他のポイント*」の「*スタック*」（『*プログラミングガイド*』）を参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												
operand1	C	S	A	G	N	A												
operand2	C	S	A	G	N	A	U	N	P	I	F	B	D	T	L		G	

ステートメント参照	ダイナミック定義
可	可
可	可

構文要素の説明：

TOP	TOP を指定した場合、データ／プログラム／コマンドが Natural スタックの先頭に挿入されます。 それ以外の場合、スタックの最後に挿入されます。

	例：次のステートメントでは、変数 #FIELDA の内容がデータとしてスタックの先頭に挿入されます。 `STACK TOP #FIELDA`
DATA	DATA（デフォルト）では、スタックに挿入されたデータは INPUT ステートメントの入力データとして使用されます。 データ内の区切り文字または INPUT 割り当て文字はデリミタとして処理されます。 スタックのデータが INPUT ステートメントでどのように処理されるかについては、「*Natural スタックデータの処理*」（INPUT ステートメントの説明内）を参照してください。 例：次のステートメントでは、変数 #FIELD1 および #FIELD2 の内容がスタックに挿入されます。 `MOVE 'ABC' TO #FIELD1` `MOVE 'XYZ' TO #FIELD2` `STACK #FIELD1 #FIELD2` これらの変数は、デリミタモードで Natural プログラム内の次の INPUT ステートメントにデータとして渡されます。 `INPUT #FIELD1  #FIELD2` **注意:** *operand2* が時刻変数（フォーマット T）の場合は、変数内容のうち時刻コンポーネントのみがスタックに挿入され、日付コンポーネントは挿入されません。
FORMATTED	FORMATTED では、フィールド単位ですべてのデータを次の INPUT ステートメントに渡します。キー割り当てやデリミタ文字は解釈されません。 例： 次のステートメントでは、"ABC,DEF" が #FIELD1 に、"XYZ" が #FIELD2 に挿入されます。 `MOVE 'ABC,DEF' TO #FIELD1` `MOVE 'XYZ'     TO #FIELD2` `STACK TOP DATA FORMATTED #FIELD1 #FIELD2`

	`` ... INPUT #FIELD1 #FIELD2 ``` INPUT 区切り文字をコンマ（プロファイル／セッションパラメータ ID=,）にすると、次のステートメント（キーワード FORMATTED なし）により、"ABC" は #FIELD1 に、"DEF" は #FIELD2 に挿入されます。 ``` MOVE 'ABC,DEF' TO #FIELD1 STACK TOP DATA #FIELD1 ... INPUT #FIELD1 #FIELD2 ``` **注意:** 渡されるデータにデリミタ文字、制御文字、または DBCS 文字が含まれている場合、これらの文字が不適切に解釈されないように FORMATTED オプションを使用してください。
COMMAND *operand1*	コマンド（またはプログラム名）をスタックに挿入するには、コマンド（*operand1*）をキーワード COMMAND に続けて指定します。 Natural は、NEXT プロンプトを表示してユーザーに入力を促す代わりに、このコマンドを実行します。 例：次のステートメントでは、コマンド RUN がスタックの先頭に挿入されます。 通常では NEXT プロンプトが発行される時点で、Natural はこのコマンドを実行します。 ``` STACK TOP COMMAND 'RUN' ```
COMMAND *operand1* *operand2* **...**	コマンド（*operand1*）とともにデータ（*operand2*）をスタックに挿入することもできます。 これらのデータは、コマンド実行後の次の INPUT ステートメントで処理されます。 コマンドとともにスタックに挿入されたデータは常にフォーマットされません。 **注意:** スタックに挿入されるデータに空の英数字フィールド（つまり、空白）が含まれている場合、これらの空白は値と値の間のデリミタとして解釈され、対応する INPUT ステートメントで正しく処理されません。そのため、コマンドとともに空の英数字フィールドをデータとしてスタックに挿入する場合、STACK ステートメントを 2 つ使用する必要があります。1 つはデータをスタックに挿入するための STACK DATA *operand2* ...、もう 1 つはコマンドをスタックに挿入するための STACK COMMAND *operand1* です。
parameter	*operand2* が日付変数の場合、この変数に対するパラメータとして、セッションパラメータ DF を指定できます。

例

```
** Example 'STKEX1': STACK
************************************************************************
DEFINE DATA LOCAL
1 #CODE (A1)
END-DEFINE
*
INPUT //
  10X 'PLEASE SELECT COMMAND' //
  10X 'LIST VIEW      (V)' /
  10X 'LIST PROGRAM * (P)' /
  10X 'TECH INFO      (T)' /
  10X 'STOP           (.)' //
  20X 'CODE:' #CODE
*
*
DECIDE ON FIRST #CODE
  VALUE 'V'
    STACK TOP DATA    'VIEW'
    STACK TOP COMMAND 'LIST'
  VALUE 'P'
    STACK TOP COMMAND 'LIST PROGRAM *'
  VALUE 'T'
    STACK TOP COMMAND 'LAST *'
    STACK TOP COMMAND 'TECH'
    STACK TOP COMMAND 'SYSPROD'
  VALUE '.'
    STOP
  NONE
    REINPUT 'PLEASE ENTER VALID CODE'
END-DECIDE
*
*
END
```

プログラム **STKEX1** の出力：

```
PLEASE SELECT COMMAND

LIST VIEW      (V)
LIST PROGRAM * (P)
TECH INFO      (T)
STOP           (.)
```

```
          CODE:P
```

コードを入力して確認した後：

```
16:46:28                ***** NATURAL LIST COMMAND *****              2005-01-19
User HTR                  - LIST Objects in a Library -          Library SYSEXSYN

Cmd  Name       Type          S/C  SM Version  User ID    Date        Time
---  *_________ P__________  *__  *  *______  *________ *__________ *________
__   ACREX1     Program       S/C  S  4.1.03   RKE        2004-11-11  16:32:37
__   ACREX2     Program       S/C  S  4.1.03   RKE        2005-01-05  10:29:51
__   ADDEX1     Program       S/C  S  4.1.03   RKE        2004-11-11  16:36:49
__   AEDEX1R    Program       S/C  R  4.1.03   RKE        2004-11-11  16:40:34
__   AEDEX1S    Program       S/C  S  4.1.03   RKE        2004-11-11  16:39:57
__   AEPEX1R    Program       S/C  R  4.1.03   RKE        2004-11-11  16:41:57
__   AEPEX1S    Program       S/C  S  4.1.03   RKE        2004-11-11  16:42:31
__   AEPEX2     Program       S/C  S  4.1.03   RKE        2004-11-11  16:43:37
__   ASDEX1R    Program       S/C  R  4.1.03   RKE        2004-11-11  17:00:21
__   ASDEX1S    Program       S/C  S  4.1.03   RKE        2004-11-11  17:00:50
__   ASGEX1R    Program       S/C  R  4.1.03   RKE        2004-11-11  17:02:01
__   ASGEX1S    Program       S/C  S  4.1.03   RKE        2004-11-11  17:02:08
__   ATBEX1R    Program       S/C  R  4.1.03   RKE        2004-11-11  17:03:18
__   ATBEX1S    Program       S/C  S  4.1.03   RKE        2004-11-11  17:03:05
                                                                 14 Objects found
Top of List.
Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Print Exit  Sort        --    -     +     ++          >     Canc
```

127 STOP

STOP

このchapterでは、次のトピックについて説明します。

機能

STOP ステートメントは、プログラムの実行を終了させ、コマンド入力プロンプトに戻すために使用します。

1 つ以上の STOP ステートメントを Natural プログラム内の任意の位置に挿入できます。

STOP ステートメントでは、プログラムの実行を即座に終了させます。 サブルーチン内の STOP ステートメントの位置に関係なく、メインプログラムで指定されたページ終了条件は、STOP ステートメントの実行中に最後のページ終了処理のために呼び出されます。

Natural RPC については、『*Natural リモートプロシージャコール（RPC）*』ドキュメントの「*サーバーに対する Natural ステートメントの注意事項*」を参照してください。

例

```
** Example 'STPEX1': STOP
************************************************************************
DEFINE DATA LOCAL
1 #CODE (A1)
END-DEFINE
*
INPUT //
  10X 'PLEASE SELECT COMMAND' //
  10X 'LIST VIEW      (V)' /
  10X 'LIST PROGRAM * (P)' /
  10X 'TECH INFO      (T)' /
  10X 'STOP           (.)' //
  20X 'CODE:' #CODE
*
*
DECIDE ON FIRST #CODE
  VALUE 'V'
    STACK TOP DATA    'VIEW'
    STACK TOP COMMAND 'LIST'
  VALUE 'P'
    STACK TOP COMMAND 'LIST PROGRAM *'
  VALUE 'T'
    STACK TOP COMMAND 'LAST *'
    STACK TOP COMMAND 'TECH'
```

```
    STACK TOP COMMAND 'SYSPROD'
  VALUE '.'
    STOP
  NONE
    REINPUT 'PLEASE ENTER VALID CODE'
END-DECIDE
*
*
END
```

プログラム **STPEX1** の出力：

```

PLEASE SELECT COMMAND

LIST VIEW      (V)
LIST PROGRAM * (P)
TECH INFO      (T)
STOP           (.)

          CODE:
```

128 STORE

ストラクチャードモード構文

```
STORE   [RECORD] [IN] [FILE] view-name
        [PASSWORD=operand1]
        [CIPHER=operand2]
        [       [       USING  ]  NUMBER operand3  ]
                [       GIVING ]
```

レポーティングモード構文

```
STORE   [RECORD] [IN] [FILE] view-name
        [PASSWORD=operand1]
        [CIPHER=operand2]
        [       [       USING  ]          NUMBER operand3  ]
                [       GIVING ]
        {       [USING] SAME [RECORD] [AS] [STATEMENT [(r)]]
                [       SET    ]          [operand4=operand5]
                [       WITH   ]          ...              }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **UPDATE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

STORE ステートメントは、データベースにレコードを追加するために使用します。

データベース固有の考慮事項

Adabas	Natural システム変数 *ISN には、STORE ステートメントの実行の結果として新しいレコードに割り当てられた Adabas ISN が含まれます。 *ISN への後続の参照には、関連する STORE ステートメントのステートメント番号が含まれている必要があります。
SQL	このステートメントは、SQL テーブルに行を追加するために使用できます。PASSWORD 節、CIPHER 節、および GIVING NUMBER 節は使用できません。 STORE ステートメントは、SQL の INSERT ステートメントに相当します。 Natural システム変数 *ISN は使用できません。
XML	このステートメントは、データベースに XML オブジェクトを追加するために使用できます。PASSWORD 節、CIPHER 節、および GIVING NUMBER 節は使用できません。 Tamino の場合、Natural システム変数 *ISN には、STORE ステートメントの実行の結果として新しいレコードに割り当てられた XML オブジェクト ID が含まれます。 *ISN への後続の参照には、関連する STORE ステートメントのステートメント番号が含まれている必要があります。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S				A												可	不可
operand2	C	S						N										可	不可
operand3		S						N	P			B *						不可	可
operand4		S	A			A	U	N	P	I	F	B	D	T	L			不可	不可
operand5	C	S	A			A	U	N	P	I	F	B	D	T	L			可	不可

* *operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

view-name	*view-name* として、ビューの名前を指定します。この名前は、DEFINE DATA ステートメント内、あるいはプログラム外のグローバルデータエリアまたはローカルデータエリアに事前に定義しておく必要があります。 レポーティングモードでは、DEFINE DATA ステートメントを使用しない場合、*view-name* には DDM の名前も使用できます。
PASSWORD=*operand1*	PASSWORD 節は、Adabas データベースに対してのみ適用可能です。

	この節は、パスワード保護されたファイルのデータを更新するときのパスワード（*operand1*）の指定に使用します。パスワード（*operand1*）は、英数字定数または英数字変数として指定できます。 最大 8 文字から成り、特殊文字または埋め込み空白は使用できません。パスワードを定数として指定する場合は、一重引用符で囲む必要があります。 詳細については、FIND ステートメントおよび PASSW ステートメントを参照してください。
CIPHER=*operand2*	CIPHER 節は、Adabas データベースに対してのみ適用可能です。 この節は、暗号化されたファイルのデータを更新するときのサイファキー（*operand2*）の指定に使用します。 サイファキー（*operand2*）は、数値定数（8 桁）またはユーザー定義変数（フォーマット／長さ N8）として指定できます。 詳細については、FIND ステートメントを参照してください。
USING NUMBER *operand3* **または** **GIVING NUMBER** *operand3*	この節は、Adabas データベースでのみ使用できます。 この節は、ユーザー指定の Adabas ISN を持つレコードを保存するために使用します。 指定した ISN を持つレコードがすでに存在しているときには、エラーメッセージが返され、ON ERROR 処理の指定がない場合、プログラムの実行は終了します。
SET/WITH *operand4*=*operand5*	SET/WITH は、レポーティングモードで値を与えるフィールドを指定するために使用します ファイルに定義されているフィールドで、SET 節で指定されないフィールドは、新しいレコード内で空値となります。 この節は、DEFINE DATA ステートメントを使用している場合は許可されません。STORE ステートメントは DEFINE DATA ステートメント内の定義に従って常にビュー全体を参照するからです。
USING SAME (*r***)**	USING SAME は、レポーティングモードで使用され、STORE ステートメントが参照するステートメント（FIND、GET、READ）で読み込まれたフィールドと同じフィールドが新しいレコードとして追加されることを示します。 ステートメント参照表記 (*r*) は、ソースコード行番号またはステートメントラベルとして指定できます。 この節は、DEFINE DATA ステートメントを使用している場合は許可されません。STORE ステートメントは DEFINE DATA ステートメント内の定義に従って常にビュー全体を参照するからです。

例

```
** Example 'STOEX1S': STORE  (structured mode)
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 MAR-STAT
  2 BIRTH
  2 CITY
  2 COUNTRY
*
1 #PERSONNEL-ID (A8)
1 #NAME         (A20)
1 #FIRST-NAME   (A15)
1 #BIRTH-D      (D)
1 #MAR-STAT     (A1)
1 #BIRTH        (A8)
1 #CITY         (A20)
1 #COUNTRY      (A3)
1 #CONF         (A1)
END-DEFINE
*
REPEAT
  INPUT 'ENTER A PERSONNEL ID AND NAME (OR ''END'' TO END)' //
        'PERSONNEL-ID : ' #PERSONNEL-ID  //
        'NAME         : ' #NAME          /
        'FIRST-NAME   : ' #FIRST-NAME
  /*
  /*  VALIDATE ENTERED DATA
  /*
  IF #PERSONNEL-ID = 'END' OR #NAME = 'END'
    STOP
  END-IF
  IF #NAME = ' '
    REINPUT WITH TEXT 'ENTER A LAST-NAME' MARK 2 AND SOUND ALARM
  END-IF
  IF #FIRST-NAME = ' '
    REINPUT WITH TEXT 'ENTER A FIRST-NAME' MARK 3 AND SOUND ALARM
  END-IF
  /*
  /*  ENSURE PERSON IS NOT ALREADY ON FILE
  /*
  FIND NUMBER EMPL-VIEW WITH PERSONNEL-ID =  #PERSONNEL-ID
```

```
IF *NUMBER > 0
  REINPUT 'PERSON WITH SAME PERSONNEL-ID ALREADY EXISTS'
           MARK 1 AND SOUND ALARM
END-IF
MOVE 'N' TO #CONF
/*
/*  GET FURTHER INFORMATION
/*
INPUT
  'ADDITIONAL PERSONNEL DATA'                          ////
  'PERSONNEL-ID             :' #PERSONNEL-ID (AD=IO) /
  'NAME                     :' #NAME         (AD=IO) /
  'FIRST-NAME               :' #FIRST-NAME   (AD=IO) ///
  'MARITAL STATUS           :' #MAR-STAT             /
  'DATE OF BIRTH (YYYYMMDD) :' #BIRTH                /
  'CITY                     :' #CITY                 /
  'COUNTRY (3 CHARACTERS)   :' #COUNTRY              //
  'ADD THIS RECORD (Y/N)    :' #CONF          (AD=M)
/*
/*   ENSURE REQUIRED FIELDS CONTAIN VALID DATA
/*
IF NOT (#MAR-STAT = 'S'  OR = 'M' OR = 'D' OR = 'W')
  REINPUT TEXT 'ENTER VALID MARITAL STATUS  S=SINGLE ' -
               'M=MARRIED D=DIVORCED W=WIDOWED' MARK 1
END-IF
IF NOT (#BIRTH = MASK(YYYYMMDD) AND #BIRTH = MASK(1582-2699))
  REINPUT TEXT 'ENTER CORRECT DATE' MARK 2
END-IF
IF #CITY  = ' '
  REINPUT TEXT 'ENTER A CITY NAME' MARK 3
END-IF
IF #COUNTRY = ' '
  REINPUT TEXT 'ENTER A COUNTRY CODE' MARK 4
END-IF
IF NOT (#CONF = 'N' OR= 'Y')
  REINPUT TEXT 'ENTER Y (YES) OR N (NO)' MARK 5
END-IF
IF #CONF = 'N'
  ESCAPE TOP
END-IF
/*
/*  ADD THE RECORD
/*
MOVE EDITED #BIRTH TO #BIRTH-D (EM=YYYYMMDD)
/*
EMPL-VIEW.PERSONNEL-ID := #PERSONNEL-ID
EMPL-VIEW.NAME         := #NAME
EMPL-VIEW.FIRST-NAME   := #FIRST-NAME
EMPL-VIEW.MAR-STAT     := #MAR-STAT
EMPL-VIEW.BIRTH        := #BIRTH-D
EMPL-VIEW.CITY         := #CITY
EMPL-VIEW.COUNTRY      := #COUNTRY
```

```
  /*
  STORE RECORD IN EMPL-VIEW
  /*
  END OF TRANSACTION
  /*
  WRITE NOTITLE 'RECORD HAS BEEN ADDED'
  /*
END-REPEAT
END
```

プログラム **STOEX1S** の出力：

```
ENTER A PERSONNEL ID AND NAME (OR 'END' TO END)

PERSONNEL-ID : 90001100

NAME         : JONES
FIRST-NAME   : EDWARD
```

人事キーデータを入力および確定した後、追加の人事データフィールドが入力用に表示されます。

```
ADDITIONAL PERSONNEL DATA



PERSONNEL-ID             : 90001100
NAME                     : JONES
FIRST-NAME               : EDWARD


MARITAL STATUS           :
DATE OF BIRTH (YYYYMMDD) :
CITY                     :
COUNTRY (3 CHARACTERS)   :

ADD THIS RECORD (Y/N)    : N
```

レポーティングモードの例はライブラリ SYSEXRM のプログラム **STOEX1R** を参照してください。

129 SUBTRACT

SUBTRACT ステートメントは、複数のオペランドの値を減算するために使用します。

このchapterでは、次のトピックについて説明します。

関連ステートメント：**ADD** | **COMPRESS** | **COMPUTE** | **DIVIDE** | **EXAMINE** | **MOVE** | **MOVE ALL** | **MULTIPLY** | **RESET** | **SEPARATE**

関連機能グループ：「*算術演算とデータ移動操作*」

機能

SUBTRACT ステートメントは、複数のオペランドの値を減算するために使用します。

データベースフィールドを結果フィールドとして使用した場合、SUBTRACT 演算ではプログラム内で使用される内部値のみが更新されます。 データベース内のフィールドの値は変更されないまま維持されます。

構文説明

構文 1 - SUBTRACT

```
SUBTRACT [ROUNDED] operand1 ... FROM operand2
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット										ステートメント参照	ダイナミック定義
operand1	C	S	A		N		N	P	I	F		D	T				可	不可
operand2		S	A		M		N	P	I	F		D	T				可	不可

構文要素の説明：

operand1 **FROM** *operand2*	オペランド： *operand1*は被減数、*operand2*は減数です。したがって、ステートメントは次と等しくなります。 *<oper2>* := *<oper2>* - *<oper1>* オペランドのフォーマットについては、「*算術割り当てのルール*」の「*フォーマット混合式のパフォーマンスについて*」（『*プログラミングガイド*』）も参照してください。

ROUNDED	切り上げ： キーワード ROUNDED を指定すると、結果は切り上げられます。切り上げについては、『*プログラミングガイド*』の「*演算割り当てのルール*」、「*フィールドの切り捨てと切り上げ*」を参照してください。

構文 2

```
SUBTRACT [ROUNDED] operand1 ... FROM operand2 GIVING operand3
```

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1	C	S	A		N			N	P	I	F		D	T					可	不可
operand2	C	S	A		N			N	P	I	F		D	T					可	不可
operand3		S	A		M	A	U	N	P	I	F	B*	D	T					可	可

**operand3* のフォーマット B は、4 以下の長さでのみ使用できます。

構文要素の説明：

GIVING	結果フィールド： GIVING 節を使用すると、*operand2* は変更されず、*operand3* に結果が入ります。
operand1 **FROM** *operand2* **GIVING** *operand3*	オペランド： *operand2* は被減数、*operand1* は減数、*operand3* は結果フィールドです。したがって、ステートメントは次と等しくなります。 *<oper3>* := *<oper2>*- *<oper1>* オペランドのフォーマットについては、「フォーマット混合式のパフォーマンスについて」（『*プログラミングガイド*』）も参照してください。
ROUNDED	切り上げ： キーワード ROUNDED を指定すると、結果は切り上げられます。 切り上げについては、『*プログラミングガイド*』の「*演算割り当てのルール*」、「*フィールドの切り捨てと切り上げ*」を参照してください。

例

```
** Example 'SUBEX1': SUBTRACT
************************************************************************
DEFINE DATA LOCAL
1 #A (P2) INIT <50>
1 #B (P2)
1 #C (P1.1) INIT <2.4>
END-DEFINE
*
SUBTRACT 6 FROM #A
WRITE NOTITLE 'SUBTRACT 6 FROM #A              '  10X '=' #A
*
SUBTRACT 6 FROM 11 GIVING #A
WRITE         'SUBTRACT 6 FROM 11 GIVING #A    '  10X '=' #A
*
SUBTRACT 3 4 FROM #A GIVING #B
WRITE         'SUBTRACT 3 4 FROM #A GIVING #B  '  10X '=' #A '=' #B
*
SUBTRACT -3 -4 FROM #A GIVING #B
WRITE         'SUBTRACT -3 -4 FROM #A GIVING #B'  10X '=' #A '=' #B
*
SUBTRACT ROUNDED 2.06 FROM #C
WRITE         'SUBTRACT ROUNDED 2.06 FROM #C   '  10X '=' #C
*
END
```

プログラム **SUBEX1** の出力：

```
SUBTRACT 6 FROM #A                       #A:  44
SUBTRACT 6 FROM 11 GIVING #A             #A:   5
SUBTRACT 3 4 FROM #A GIVING #B           #A:   5 #B:  -2
SUBTRACT -3 -4 FROM #A GIVING #B         #A:   5 #B:  12
SUBTRACT ROUNDED 2.06 FROM #C            #C:  0.3
```

130 SUSPEND IDENTICAL SUPPRESS

```
SUSPEND IDENTICAL [SUPPRESS] [(rep)]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **WRITE** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

SUSPEND IDENTICAL SUPPRESS ステートメントは、1 レコードの処理に対して（同一フィールド値の出力を省略する）Natural セッションパラメータ設定 IS=ON を中止するために使用します。

セッションパラメータ IS（『*パラメータリファレンス*』）も参照してください。

構文説明

(*rep*)	レポート指定： 表記 (*rep*) を使用して、SUSPEND IDENTICAL SUPPRESS ステートメントを適用するレポートの ID を指定できます。 範囲 0～31 の値、または DEFINE PRINTER ステートメントを使用して割り当てた論理名を指定できます。 (*rep*) 指定がない場合、SUSPEND IDENTICAL SUPPRESS ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。

例

例 1 - SUSPEND IDENTICAL SUPPRESS を使用したプログラム

```
** Example 'SISEX1': SUSPEND IDENTICAL SUPPRESS
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
*
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  /*
  SUSPEND IDENTICAL SUPPRESS
  /*
  FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      MOVE '***NO CAR***' TO MAKE
    END-NOREC
    DISPLAY NOTITLE
            NAME (RD.) (IS=ON)
            FIRST-NAME (RD.) (IS=ON)
            MAKE (FD.)
  END-FIND
  /*
END-READ
END
```

プログラム **SISEX1** の出力：

```
        NAME             FIRST-NAME             MAKE
-------------------- -------------------- --------------------

JONES                VIRGINIA             CHRYSLER
JONES                MARSHA               CHRYSLER
                                          CHRYSLER
JONES                ROBERT               GENERAL MOTORS
JONES                LILLY                FORD
                                          MG
JONES                EDWARD               GENERAL MOTORS
JONES                MARTHA               GENERAL MOTORS
JONES                LAUREL               GENERAL MOTORS
JONES                KEVIN                DATSUN
JONES                GREGORY              FORD
JONES                EDWARD               ***NO CAR***
JOPER                MANFRED              ***NO CAR***
JOUSSELIN            DANIEL               RENAULT
JUBE                 GABRIEL              ***NO CAR***
JUNG                 ERNST                ***NO CAR***
JUNKIN               JEREMY               ***NO CAR***
```

例 2 - 前のプログラムと同様（ただし、SUSPEND IDENTICAL SUPPRESS は使用しない）

```
** Example 'SISEX2': SUSPEND IDENTICAL SUPPRESS (compare with SISEX1)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  /*
  /* SUSPEND IDENTICAL SUPPRESS    /* statement removed
  /*
  FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      MOVE '***NO CAR***' TO MAKE
    END-NOREC
    DISPLAY NOTITLE
            NAME (RD.) (IS=ON)
            FIRST-NAME (RD.) (IS=ON)
            MAKE (FD.)
```

```
  END-FIND
  /*
END-READ
END
```

プログラム **SISEX2** の出力：

```
        NAME             FIRST-NAME             MAKE
-------------------- -------------------- --------------------

JONES                VIRGINIA             CHRYSLER
                     MARSHA               CHRYSLER
                                          CHRYSLER
                     ROBERT               GENERAL MOTORS
                     LILLY                FORD
                                          MG
                     EDWARD               GENERAL MOTORS
                     MARTHA               GENERAL MOTORS
                     LAUREL               GENERAL MOTORS
                     KEVIN                DATSUN
                     GREGORY              FORD
                     EDWARD               ***NO CAR***
JOPER                MANFRED              ***NO CAR***
JOUSSELIN            DANIEL               RENAULT
JUBE                 GABRIEL              ***NO CAR***
JUNG                 ERNST                ***NO CAR***
JUNKIN               JEREMY               ***NO CAR***
```

131 TERMINATE

```
TERMINATE [operand1 [operand2]]
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

TERMINATE ステートメントは、Natural セッションを終了するために使用します。 TERMINATE ステートメントは、Natural プログラム内の任意の位置に指定できます。 TERMINATE ステートメントが実行されると、ページ終了処理やループ終了処理は行われません。

Natural RPC については、『*Natural リモートプロシージャコール（RPC）*』ドキュメントの「*サーバーに対する Natural ステートメントの注意事項*」を参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット										ステートメント参照	ダイナミック定義
operand1	C	S					N	P	I							可	不可
operand2	C	S	A			A	U									可	可

構文要素の説明：

operand1	*operand1* を使用して、Natural 終了時に制御を受け取るプログラムにリターンコードを渡します。例えば、リターンコード設定は、コマンドプロセッサに渡し、ERRORLEVEL 機能でチェックできます。渡すことができます。 『*オペレーション*』ドキュメントの「*Natural スタートアップエラー*」も参照してください。 *operand1* に指定できる値の範囲は 0～255 です。
operand2	*operand2* を使用して、終了後に制御を受け取るプログラムに追加情報を渡すことができます。

Natural 終了後に制御を受け取るプログラム

Naturalセッションの終了後、プロファイルパラメータ PROGRAM で指定された名前のプログラムに制御が渡ります。

Naturalは、そのプログラムに*operand2*とプロファイルパラメータ PRGPAR の値を渡します（これらが指定されている場合）。 プログラムはこれらのパラメータを通常の方法で引数として受け取ります。

```
int main(int argc, char *argv[])
{
  /* Number of arguments passed. */
  printf("Number of arguments: %d\n", argc);
  /* Program name. */
  if ( argc > 0 )
    printf("Program: %s\n", argv[0]);
  /* Value of operand2 of the TERMINATE statement. */
  if ( argc > 1 )
    printf("Operand 2: %s\n", argv[1]);
  /* Value of the profile parameter PRGPAR. */
  if ( argc > 2 )
    printf("PRGPAR: %s\n", argv[2]);
  return 0;
}
```

PROGRAMパラメータが設定されていない場合、終了後、コマンドインタープリタが制御を受け取ります。

例

```
** Example 'TEREX1': TERMINATE
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 SALARY (1)
*
1 #PNUM     (A8)
1 #PASSWORD (A8)
END-DEFINE
*
INPUT 'ENTER PASSWORD:' #PASSWORD
```

```
*
IF #PASSWORD NE 'USERPASS'
  /*
  TERMINATE
  /*
END-IF
*
INPUT 'ENTER PERSONNEL NUMBER:' #PNUM
*
FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PNUM
  DISPLAY NAME SALARY (1)
END-FIND
*
END
```

132 UPDATE

ストラクチャードモード構文

```
UPDATE [RECORD] [IN] [STATEMENT] [(r)]
```

レポーティングモード構文

```
UPDATE   [RECORD] [IN] [STATEMENT] [(r)]

         [          SET          ]
         [          WITH         ] { SAME [RECORD]                }
         [          USING        ] { {operand1=operand2} ...      }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**ACCEPT/REJECT** | **AT BREAK** | **AT START OF DATA** | **AT END OF DATA** | **BACKOUT TRANSACTION** | **BEFORE BREAK PROCESSING** | **DELETE** | **END TRANSACTION** | **FIND** | **GET** | **GET SAME** | **GET TRANSACTION DATA** | **HISTOGRAM** | **LIMIT** | **PASSW** | **PERFORM BREAK PROCESSING** | **READ** | **RETRY** | **STORE**

関連機能グループ：「*データベースへのアクセスと更新*」

機能

UPDATE ステートメントは、データベースレコードの 1 つ以上のフィールドを更新するために使用します。 更新するレコードは、FIND、GET、READ（または Adabas の場合は STORE）ステートメントで事前に選択されている必要があります。

ホールド状態

UPDATE ステートメントの使用により、対応する FIND や READ ステートメントで選択された各レコードはホールド状態になります。

詳細については、「*レコードホールドロジック*（『*プログラミングガイド*』）を参照してください。

制限事項

- UPDATE ステートメントを、更新するレコードを選択するステートメントと同一行に入力しないでください。
- UPDATE ステートメントは Entire System Server ビューには適用できません。

データベース固有の考慮事項

SQL	UPDATE ステートメントは、データベーステーブルの行を更新するために使用できます。 このステートメントは SQL の UPDATE WHERE CURRENT OF CURSOR（位置決め UPDATE）に相当し、最後に読み込まれた行のみを更新できます。 ほとんどの SQL データベースで、FIND SORTED BY ステートメントや READ LOGICAL ステートメントによって読み込まれた行は更新できません。
XML	XML データベースでは UPDATE ステートメントを使用できません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1		S	A			A		N	P	I	F	B	D	T	L				不可	不可
operand2	C	S	A			A		N	P	I	F	B	D	T	L				可	不可

構文要素の説明：

(*r*)	ステートメント参照： 表記（*r*）を使用して、更新するレコードが選択されたステートメントを示すことができます。 *r* は、ソースコード行番号またはステートメントラベルとして指定できます。 参照指定がなければ、UPDATE ステートメントでは、一番内側のアクティブな READ または FIND 処理ループが参照されます。 アクティブな READ または FIND ループがない場合、このステートメントより前の最後の GET（または STORE）ステートメントが参照されます。 **注意:** UPDATE ステートメントは、参照している READ または FIND ループ内に指定する必要があります。

USING SAME	この節は、DEFINE DATA ステートメントを使用している場合は許可されません。UPDATE ステートメントは DEFINE DATA ステートメント内の定義に従って常にビュー全体を参照するからです。 レコードバッファまたはフォーマットバッファのレイアウトは、OBTAIN ステートメントを使用して宣言できます。 USING SAME は、レポーティングモードで UPDATE ステートメントが参照するステートメントで読み込まれたフィールドと同じフィールドを更新に使用することを示します。 この場合、各データベースフィールドに割り当てられた最新の値がフィールドの更新に使用されます。 新しい値の割り当てがなければ、古い値が使用されます。 更新するフィールドがマルチプルバリューフィールドやピリオディックグループの配列範囲で、この配列範囲に変数インデックスを使用する場合、最新の範囲が更新されます。つまり、レコードが読み込まれた後で、かつUPDATE USING SAME（レポーティングモード）またはUPDATE（ストラクチャードモード）の各ステートメントの実行前にインデックス変数が変更されると、更新される範囲は読み込み時の範囲と同じではなくなります。
SET/WITH *operand1*=*operand2*	この節は、レポーティングモードで更新するフィールドと値を指定するために使用します。 この節は、DEFINE DATA ステートメントを使用している場合は許可されません。UPDATE ステートメントは DEFINE DATA ステートメント内の定義に従って常にビュー全体を参照するからです。

例

```
** Example 'UPDEX1S': UPDATE (structured mode)
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
*
1 #NAME (A20)
END-DEFINE
*
INPUT 'ENTER A NAME:' #NAME (AD=M)
IF #NAME = ' '
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH NAME = #NAME
```

```
  IF NO RECORDS FOUND
    REINPUT WITH 'NO RECORDS FOUND'  MARK 1
  END-NOREC
  INPUT 'NAME:      ' NAME (AD=O) /
        'FIRST NAME:' FIRST-NAME (AD=M) /
        'CITY:      ' CITY (AD=M)
  UPDATE
  END TRANSACTION
END-FIND
*
END
```

プログラム **SUBEX1S** の出力：

```
ENTER A NAME: BROWN
```

名前を入力して確認した後：

```
NAME: BROWN
FIRST NAME: KENNETH
CITY: DERBY
```

レポーティングモードの例はライブラリ SYSEXRM のプログラム **UPDEX1R** を参照してください。

133 WRITE

WRITE ステートメントは、出力をフリーフォーマットで生成するために使用します。

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE TITLE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

WRITE ステートメントは、出力をフリーフォーマットで生成するために使用します。

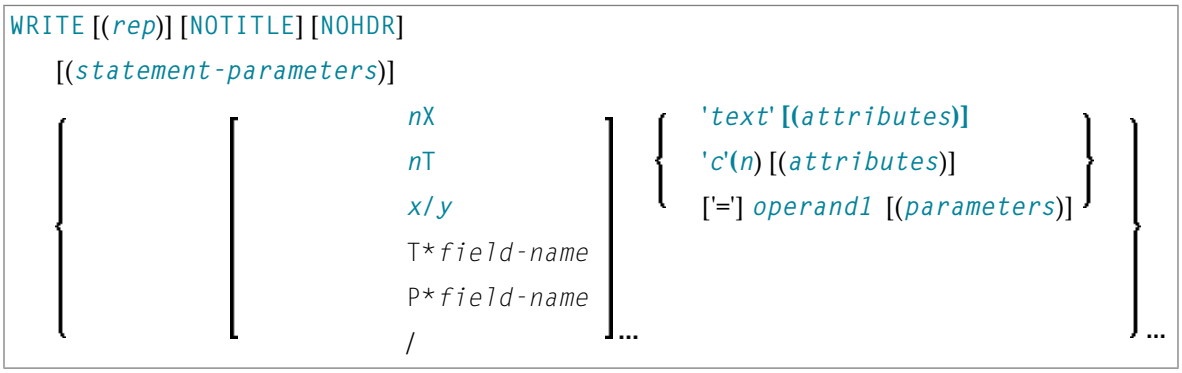
WRITE ステートメントは、次の点で DISPLAY ステートメントと異なります。

- 行あふれをサポートします。 レポートの幅が 1 行を超過する場合、あふれたフィールド（またはテキスト）は次の行に書き出されます。 フィールドやテキストが行をまたがって出力されることはありません。
- デフォルトの列ヘッダーは出力されません。 データの長さによって各フィールドの出力桁数が決まります。
- 配列の値やオカレンスは縦出力ではなく、横に並べられます。

『*プログラミングガイド*』の次のトピックも参照してください。

- *データ出力制御*
- *DISPLAY および WRITE ステートメント*
- *マルチプルバリューフィールドとピリオディックグループのインデックス表記*
- *DISPLAY VERT と WRITE ステートメントの例*
- *出力ページのレイアウト*

構文1 - 直接画面定義



構文図で使用されている記号については、「*構文記号*」を参照してください。

構文 1 - 説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1		S	A	G	N	A	U	N	P	I	F	B	D	T	L		G	O	可	不可

構文要素の説明：

(*rep*)	レポート指定： プログラムで複数のレポートが生成される場合に、表記（*rep*）を使用してレポートの ID を指定します。 レポート ID として、範囲 0～31 の値、または DEFINE PRINTER ステートメントを使用して割り当てた論理名を指定できます。 （*rep*）を指定しない場合、ステートメントは最初のレポート（レポート 0）に適用されます。 このプリンタファイルが PC として Natural に定義されている場合、レポートは PC にダウンロードされます。*例 5* を参照してください。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。
NOTITLE	デフォルトページタイトルの省略：

	Natural は、WRITE ステートメントによって出力される各ページに 1 行のタイトル行を生成します。 このタイトルには、ページ番号と日時が含まれます。 時刻はプログラム実行の開始時に設定されます。 このデフォルトのタイトル行は WRITE TITLE ステートメントを使用して上書きできます。また、WRITE ステートメントに NOTITLE オプションを指定することにより省略することもできます。 例： ■ デフォルトタイトルが生成されます。 `WRITE NAME` ■ ユーザータイトルが生成されます。 `WRITE NAME WRITE TITLE 'user-title'` ■ タイトルは生成されません。 `WRITE NOTITLE NAME` **注意:** 1. NOTITLE オプションを使用すると、同じオブジェクト内でデータを同じレポートに書き込むすべての DISPLAY、PRINT、および WRITE ステートメントに適用されます。 2. WRITE ステートメントの実行*前*にページオーバーフローがチェックされます。 WRITE ステートメントの実行*中*にタイトルまたはトレイラ情報付きの新しいページが生成されることはありません。
NOHDR	列ヘッダーの省略： WRITE ステートメント自体では、列ヘッダーは生成されません。 ただし、WRITE ステートメントを DISPLAY ステートメントと組み合わせて使用する場合、WRITE ステートメントの NOHDR オプションを使用して、DISPLAY ステートメントによって生成される列ヘッダーを省略できます。 NOHDR オプションは、WRITE ステートメントの実行によって新しいページが出力される場合にのみ有効です。 NOHDR オプションを使用しないと、この新しいページに DISPLAY ステートメントの列ヘッダー（存在する場合）が出力されます。NOHDR を使用すると、出力されません。
statement-parameters	ステートメントレベルでのパラメータ定義： 1つまたは複数のパラメータをカッコで囲んで、ステートメントレベルで、つまり WRITE ステートメントの直後に指定できます。

	指定した各パラメータは、以前に GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードでのみ）、または FORMAT ステートメントで指定した対応するパラメータを上書きします。 複数のパラメータを指定する場合は、個々のパラメータを1つ以上の空白で区切る必要があります。 各パラメータ指定を 2 行のステートメント行に分割することはできません。 **注意:** ここで適用されるパラメータ設定は、変数フィールドにのみ関連し、テキスト定数には影響しません。 テキスト定数にフィールド属性を設定する場合は、この要素に属性を明示的に設定する必要があります（「*要素（フィールド）レベルでのパラメータ定義*」を参照）。 以下の項目も参照してください。 ■ *パラメータのリスト* ■ *ステートメントおよび要素（フィールド）レベルでのパラメータ使用例* ■ *例 5 - ステートメント／要素（フィールド）レベルで '=' とパラメータを使用した WRITE ステートメント*
n*X**, ***n*T**, *x*/*y*, **Tfield-name*, **P***field-name*, **'='**, *l*,	フィールドの位置指定表記： 「*出力フォーマット定義*」セクションの「*フィールドの位置指定表記*」を参照してください。
'*text***'**, **'***c***'**(*n*), *attributes*, *operand1*, *parameters*	テキスト／属性割り当て： 「*出力フォーマット定義*」セクションの「*テキスト／属性割り当て*」を参照してください。

パラメータのリスト

WRITE ステートメントで指定可能なパラメータ		指定（S = ステートメントレベル、E = 要素レベル）
AD	属性定義	SE
AL	出力の英数字長	SE
CD	カラー定義	SE
CV	制御変数	SE
DF	日付フォーマット	SE
DL	出力の表示長	SE
DY	ダイナミック属性	SE
EM	編集マスク	SE
EMU	Unicode 編集マスク	E
FL	浮動小数点仮数長	SE

WRITE ステートメントで指定可能なパラメータ		指定（S = ステートメントレベル、E = 要素レベル）
IS	重複抑制	SE
LS	行サイズ	S
MC	マルチプルバリューフィールド数	S
MP	レポートの最大ページ数	S
NL	出力の数値長	SE
PC	ピリオディックグループ数	S
PM	出力モード	SE
PS	ページサイズ*	S
SG	符号の位置	SE
UC	下線付き文字	S
ZP	ゼロ出力	SE

*PS セッションパラメータ設定は、配列のオカレンス数が PS 値を超える場合は考慮されません。

各セッションパラメータの詳細については、『パラメータリファレンス』を参照してください。

『プログラミングガイド』の次のトピックも参照してください。

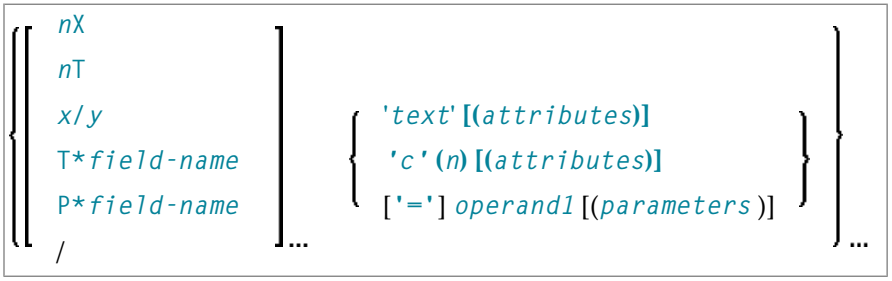
- *列ヘッダーの中央揃え - HC パラメータ*
- *列ヘッダーの幅 - HW パラメータ*
- *ヘッダーの充填文字 - FC および GC パラメータ*
- *タイトルおよびヘッダーの下線付き文字 - UC パラメータ*

ステートメントおよび要素（フィールド）レベルでのパラメータ使用例

```
DEFINE DATA LOCAL
1 VARI (A4)     INIT <'1234'>                  /*     Output
END-DEFINE                                     /*    Produced
*                                              /*    ---------
WRITE           'Text'           VARI          /*    Text 1234
WRITE (AD=U)    'Text'           VARI          /*    Text 1234
WRITE           'Text' (AD=U)    VARI (AD=U)   /*    Text 1234
WRITE           'Text' (AD=U)    VARI          /*    Text 1234
END
```

「*例 5 - ステートメント／要素（フィールド）レベルで '=' とパラメータを使用した **WRITE** ステートメント*」も参照してください。

出力フォーマット定義



構文図で使用されている記号については、「*構文記号*」を参照してください。

フィールドの位置指定表記

***n*X**	列の間隔： この表記により、列の間に *n* 個のスペースが挿入されます。 例： `WRITE NAME 5X SALARY` 以下の項目も参照してください。 ■ *例 2 - nX、nT 表記を使用した WRITE ステートメント*（後述） ■ *列の間隔 - SF パラメータと nX 表記*（『プログラミングガイド』）
***n*T**	タブ設定： *n*T 表記により、位置 *n* を出力するように位置指定（タブ設定）されます。 後方への位置指定はできません。 次の例では、NAME は位置 25 から、SALARY は位置 50 から出力されます。 `WRITE 25T NAME 50T SALARY` 以下の項目も参照してください。 ■ *例 2 - nX、nT 表記を使用した WRITE ステートメント*（後述） ■ *タブ設定 - nT 表記*（『プログラミングガイド』）
x/*y*	*x*/*y* 位置指定： *x*/*y* 表記により、次の要素は最後のステートメント出力の *x* 行下、列 *y* の先頭に配置されます。 *y* を "0" にすることはできません。 同じ行で後方に位置指定することはできません。

	「*位置指定表記 x/y*」（『*プログラミングガイド*』）も参照してください。
T*field-name*	フィールドの相対位置指定： 表記 T* では、前の DISPLAY ステートメントで使用した*フィールドの指定出力桁*に位置指定します。 後方への位置指定はできません。 以下の項目も参照してください。 ■ *例 3 - T* 表記を使用した WRITE ステートメント*（後述） ■ *タブ表記 - T*field*（『*プログラミングガイド*』）
P*field-name*	フィールドおよび行の相対位置指定： 表記 P* では、前の DISPLAY ステートメントで使用した*フィールドの指定出力桁および行*に位置指定します。 縦出力モードと組み合わせてよく使用します。 後方への位置指定はできません。 以下の項目も参照してください。 ■ *例 4 - P* 表記を使用した WRITE ステートメント*（後述） ■ *タブ表記 P*field*（『*プログラミングガイド*』）
'='	フィールドヘッダーの後に位置するフィールド内容： フィールドの前に配置した場合は、等号 '=' により、DEFINE DATA ステートメントまたは DDM の定義に従って、フィールドヘッダーはフィールド内容の前に表示されます。 以下の項目も参照してください。 ■ *例 1 - '='、'text'、'/' を使用した WRITE ステートメント* ■ *例 5 - ステートメント／要素レベルで '=' とパラメータを使用した WRITE ステートメント*
/	行送り - スラッシュ表記： フィールドまたはテキスト要素の間に配置した場合は、"/" により次の出力行の先頭に位置指定されます。 例： `WRITE NAME / SALARY` 複数の行送りをするために "/" を複数指定できます。 以下の項目も参照してください。 ■ *例 1 - '='、'text'、'/' を使用した WRITE ステートメント*（後述） ■ *行送り - スラッシュ表記*（『*プログラミングガイド*』） ■ *例 2 - WRITE ステートメントでの行送り*（『*プログラミングガイド*』）

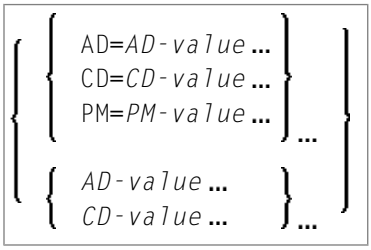
テキスト／属性割り当て

'text'	テキスト割り当て： 一重引用符で囲まれた文字列が表示されます。 例： `WRITE 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT` 以下の項目も参照してください。 ■ *例 1 - '='、'text'、'/' を使用した WRITE ステートメント*（後述） ■ 「*テキスト表記*」、「*ステートメントで使用するテキストの定義*」（『*プログラミングガイド*』）
'c'(n)	文字の繰り返し： フィールド値の直前に、一重引用符で囲まれた文字列が *n* 回表示されます。 例： `WRITE '*' (5) '=' NAME` 結果 `***** SMITH` 「*テキスト表記*」の「*フィールド値の前に n 回表示される文字の定義*」（『*プログラミングガイド*』）も参照してください。
attributes	フィールド表現と色属性： テキスト／フィールド表示にさまざまな属性を割り当てることができます。 これらの属性と構文については、以下の「*出力属性*」で説明します。 例： `WRITE 'TEXT' (BGR)` `WRITE 'TEXT' (B)` `WRITE 'TEXT' (BBLC)`
operand1	書き込まれるフィールド： *operand1* では、内容がこの位置に書き込まれるフィールドを指定します。

parameters	要素（フィールド）レベルでのパラメータ定義： 1 つまたは複数のパラメータをカッコで囲んで、要素（フィールド）レベルで、つまり *operand1* の直後に指定できます。 この方法で指定した各パラメータは、以前にステートメントレベル、GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードでのみ）、または FORMAT ステートメントで指定した対応するパラメータを上書きします。 複数のパラメータを指定する場合は、各エントリ間に 1 つ以上の空白を配置する必要があります。 エントリを 2 行のステートメント行に分割することはできません。 以下の項目も参照してください。 ■ *パラメータのリスト* ■ *ステートメントおよび要素（フィールド）レベルでのパラメータ使用例*

出力属性

attributes は、テキスト表示に使用される出力属性を示します。 可能な属性：



指定可能なセッションパラメータ値については、『パラメータリファレンス』ドキュメントの該当するセクションを参照してください。

■ 「*AD - 属性定義*」の「*フィールド表現*」
■ *CD - カラー定義*
■ *PM - 出力モード*

Note: コンパイラは、実際には 1 つの出力フィールドに複数の属性値を受け入れます。 例えば、「AD=BDI」と指定できます。 ただし、この場合は最後の値のみが適用されます。 示した例では、値 "I" のみが有効になり、出力フィールドは強調表示されます。

構文 2 - フォーム／マップ使用

```
WRITE [(rep)] [NOTITLE] [NOHDR] [USING ] { FORM } operand1 [operand2 ...]
                                         { MAP  }
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

構文 2 - 説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット														
operand1	C	S				A														
operand2		S	A	G	N	A	U	N	P	I	F	B	D	T	L					

オペランド	ステートメント参照	ダイナミック定義
operand1	不可	不可
operand2	可	不可

構文要素の説明：

FORM/MAP	定義済みフォーム／マップレイアウトの使用 このオプションでは、あらかじめ Natural マップエディタで定義しておいたフォーム／マップレイアウトを使用するように指定できます。 WRITE ステートメントで使用するマップレイアウトでは、各マップ出力ごとの自動的なページ換えは行われません。 LS パラメータはマップで定義した LS 設定よりも 1 バイト大きく指定する必要があります。
operand1	**Form/Map** 名： *operand1* は、使用するフォーム／マップの名前です。
operand2	書き込まれるフィールド： *operand2* は、出力するフィールドの名前です。 *operand1*が定数で、*operand2*が省略されている場合、コンパイル時にマップソースからフィールドが取り込まれます。 フィールドは、番号、シーケンス、フォーマット、長さ、オカレンス数（配列の場合）について、参照されているフォーム／マップに準拠している必要があります。そうでない場合、エラーが発生します。
NOTITLE/NOHDR	タイトル行／列ヘッダーの省略：

	NOTITLE および NOHDR については、WRITE ステートメントの*構文 1* を参照してください。

例

- 例 1 - '='、'*text*'、'/' を使用した WRITE ステートメント
- 例 2 - *n*X、*n*T 表記を使用した WRITE ステートメント
- 例 3 - T* 表記を使用した WRITE ステートメント
- 例 4 - P* 表記を使用した WRITE ステートメント
- 例 5 - ステートメント／要素（フィールド）レベルで '=' とパラメータを使用した WRITE ステートメント
- 例 6 - Natural に PC として定義された出力ファイルを使用したレポート指定

例 1 - '='、'*text*'、'/' を使用した WRITE ステートメント

```
** Example 'WRTEX1': WRITE (with '=', 'text', '/')
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 FULL-NAME
    3 FIRST-NAME
    3 MIDDLE-I
    3 NAME
  2 CITY
  2 COUNTRY
END-DEFINE
*
LIMIT 1
READ EMPL-VIEW BY NAME
  /*
  WRITE NOTITLE
        '=' NAME '=' FIRST-NAME '=' MIDDLE-I //
        'L O C A T I O N' /
        'CITY:   ' CITY    /
        'COUNTRY:' COUNTRY //
  /*
END-READ
END
```

プログラム **WRTEX1** の出力：

```
NAME: ABELLAN              FIRST-NAME: KEPA                MIDDLE-I:

L O C A T I O N
CITY:    MADRID
COUNTRY: E
```

例 2 - *n*X、*n*T 表記を使用した WRITE ステートメント

```
** Example 'WRTEX2': WRITE (with nX, nT notation)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY NAME
  WRITE NOTITLE 5X NAME 50T JOB-TITLE
END-READ
END
```

プログラム **WRTEX2** の出力：

```
ABELLAN                                          MAQUINISTA
ACHIESON                                         DATA BASE ADMINISTRATOR
ADAM                                             CHEF DE SERVICE
ADKINSON                                         PROGRAMMER
```

例 3 - T* 表記を使用した WRITE ステートメント

```
** Example 'WRTEX3': WRITE (with T* notation)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY (1)
END-DEFINE
*
LIMIT 5
READ EMPL-VIEW BY CITY STARTING FROM 'ALBU'
  DISPLAY NOTITLE CITY NAME SALARY (1)
  AT BREAK CITY
    /*
```

```
    WRITE / 'CITY AVERAGE:' T*SALARY (1) AVER(SALARY(1)) //
    /*
  END-BREAK
END-READ
END
```

プログラム **WRTEX3** の出力：

```
        CITY                  NAME           ANNUAL
                                             SALARY
-------------------- -------------------- ----------

ALBUQUERQUE          HAMMOND                   22000
ALBUQUERQUE          ROLLING                   34000
ALBUQUERQUE          FREEMAN                   34000
ALBUQUERQUE          LINCOLN                   41000

CITY AVERAGE:                                  32750


ALFRETON             GOLDBERG                   4800

CITY AVERAGE:                                   4800
```

例 4 - P* 表記を使用した WRITE ステートメント

```
** Example 'WRTEX4': WRITE (with P* notation)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 BIRTH
  2 SALARY (1)
END-DEFINE
*
LIMIT 3
READ EMPL-VIEW BY CITY FROM 'N'
  DISPLAY NOTITLE NAME CITY
          VERT AS 'BIRTH/SALARY' BIRTH (EM=YYYY-MM-DD) SALARY (1)
  SKIP 1
  AT BREAK CITY
    WRITE  / 'CITY AVERAGE' P*SALARY (1) AVER(SALARY (1)) //
  END-BREAK
END-READ
END
```

プログラム **WRTEX4** の出力：

```
        NAME                 CITY           BIRTH
                                            SALARY
-------------------- -------------------- ----------

WILCOX               NASHVILLE            1970-01-01
                                               38000

MORRISON             NASHVILLE            1949-07-10
                                               36000


CITY AVERAGE                                   37000


BOYER                NEMOURS              1955-11-23
                                              195900


CITY AVERAGE                                  195900
```

例 5 - ステートメント／要素（フィールド）レベルで '=' とパラメータを使用した WRITE ステートメント

```
** Example 'WRTEX5': WRITE (using '=', statement/element parameters)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
  2 PHONE
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY NAME
  WRITE NOTITLE (AL=16 NL=8)
        '=' PERSONNEL-ID '=' NAME '=' PHONE (AL=10 EM=XXX-XXXXXXX)
END-READ
END
```

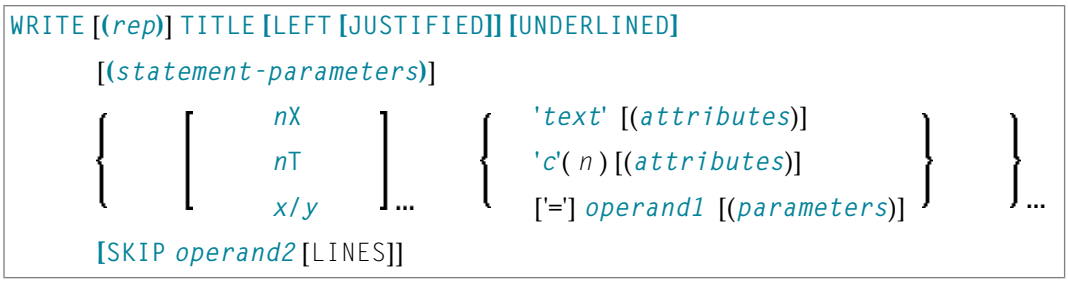
プログラム **WRTEX5** の出力：

```
PERSONNEL ID: 60008339         NAME: ABELLAN          TELEPHONE: 435-6726
PERSONNEL ID: 30000231         NAME: ACHIESON         TELEPHONE: 523-341
```

例 6 - Natural に PC として定義された出力ファイルを使用したレポート指定

```
** Example 'PCDIEX1': DISPLAY and WRITE to PC
**
** NOTE: Example requires that Natural Connection is installed.
************************************************************************
DEFINE DATA LOCAL
01 PERS VIEW OF EMPLOYEES
  02 PERSONNEL-ID
  02 NAME
  02 CITY
END-DEFINE
*
FIND PERS WITH CITY = 'NEW YORK'               /* Data selection
  WRITE (7) TITLE LEFT 'List of employees in New York' /
  DISPLAY (7)          /* (7) designates the output file (here the PC).
    'Location'  CITY
    'Surname'   NAME
    'ID'        PERSONNEL-ID
END-FIND
END
```

134 WRITE TITLE



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TRAILER**

関連機能グループ：「*出力レポートの作成*」

機能

WRITE TITLE ステートメントは、デフォルトのページタイトルではなくユーザー指定のページタイトルを表示するために使用します。 このステートメントは、新しいページが始まるたびに実行されます。

『*プログラミングガイド*』の次のセクションも参照してください。

- *データ出力制御*
- *レポート指定 - (rep) 表記*
- *出力ページのレイアウト*
- *ページタイトル、改ページ、空行*
- *独自ページタイトル定義 - WRITE TITLE ステートメント*
- *テキスト表記*

処理

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

レポートが、異なるオブジェクト内のステートメントによって作成される場合、WRITE TITLE ステートメントは、新ページを作成するステートメントと同じオブジェクトで使用されている場合に限って実行されます。

制限事項

- WRITE TITLE は 1 レポートにつき 1 回だけ指定できます。
- WRITE TITLE は特殊条件ステートメントブロック内では指定できません。
- WRITE TITLE はサブルーチン内では指定できません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1		S	A	G	N	A	U	N	P	I	F	B	D	T	L		G	O	可	不可
operand2	C	S						N	P	I		B							可	不可

構文要素の説明：

(*rep*)	レポート指定： 複数のレポートを作成するときは、表記 (*rep*) を使用して、WRITE TITLE ステートメントを適用するレポートの ID を指定できます。 レポート ID として、範囲 0～31 の値、または DEFINE PRINTER ステートメントを使用して割り当てた論理名を指定できます。 (*rep*) 指定がない場合、WRITE TITLE ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。
LEFT JUSTIFIED	桁揃えおよび下線

UNDERLINED	デフォルトでは、ページタイトルは中央揃えで下線なしです。LEFT JUSTIFIED と UNDERLINED を指定して上書きできます。 UNDERLINED を指定した場合、下線文字（デフォルトの文字または FORMAT ステートメントのセッションパラメータ UC で指定された文字）が、タイトルの下に行サイズ（セッションパラメータ LS を参照）の幅だけ出力されます。 Natural では、すべての間隔およびタブ指定を適用して行を作成してから、行全体を中央揃えします。 例えば、最初の要素としての "10T" 表記により、中央揃えされたヘッダーは右側の 5 の位置に配置されます。
statement-parameters	ステートメントレベルでのパラメータ定義： 1つまたは複数のパラメータをカッコで囲んで、ステートメントレベルで、つまり WRITE TITLE ステートメントの直後に指定できます。 この方法で指定した各パラメータは、以前に GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードでのみ）、または FORMAT ステートメントで指定したパラメータを上書きします。 複数のパラメータを指定する場合は、各エントリ間に1つ以上の空白を配置する必要があります。 エントリを 2 行のステートメント行に分割することはできません。 **注意:** ここで適用されるパラメータ設定は、変数フィールドにのみ関連し、テキスト定数には影響しません。 テキスト定数にフィールド属性を設定する場合は、この要素に属性を明示的に設定する必要があります（「*要素（フィールド）レベルでのパラメータ定義*」を参照）。 使用可能なパラメータについては、『ステートメント』ドキュメントの「WRITE」で「*パラメータのリスト*」を参照してください。
*n***X** *n***T** *x*/*y*	フォーマット表記と要素間隔： 「*フォーマット表記と要素間隔*」（後述）を参照してください。
'*text*' '*c*' (*n*) *attributes*	テキスト／属性割り当て： 「*テキスト／属性割り当て*」（後述）を参照してください。
operand1	タイトル内に表示するフィールド： *operand1* は、タイトル内に表示するフィールド（複数可）を表します。
parameters	要素（フィールド）レベルでのパラメータ定義： 1 つまたは複数のパラメータをカッコで囲んで、要素（フィールド）レベルで、つまり *operand1* の直後に指定できます。この方法で指定した各パラメータは、以前にステートメントレベル、GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードでのみ）、または FORMAT ステートメントで指定した対応するパラメータを上書きします。

	複数のパラメータを指定する場合は、各エントリ間に1つ以上の空白を配置する必要があります。 エントリを 2 行のステートメント行に分割することはできません。 使用可能なパラメータについては、『ステートメント』ドキュメントの「WRITE」で「*パラメータのリスト*」を参照してください。
SKIP *operand2* **LINES**	省略される行： SKIP を使用して、タイトル行の下に何行か空けることができます。空ける行数は数値定数または数値変数の内容で指定します。 **注意:** WRITE TITLE の後の SKIP は独立したステートメントでなく、常に WRITE TITLE ステートメントの SKIP 節として解釈されます。 WRITE TITLE ステートメントの後に独立した SKIP ステートメントが必要な場合は、セミコロン（;）を使用して 2 つのステートメントを互いに区別します。

フォーマット表記と要素間隔

*n***X**	列の間隔： この表記により、列の間に *n* 個のスペースが挿入されます。
*n***T**	タブ設定： *n*T 表記により、位置 *n* を出力するように位置指定（タブ設定）されます。 後方への位置指定はできません。
x/*y*	*x*/*y* 位置指定： 次の要素は最後のステートメント出力の *x* 行下、列 *y* の先頭に配置されます。 *y* を "0" にすることはできません。 同じ行で後方に位置指定することはできません。

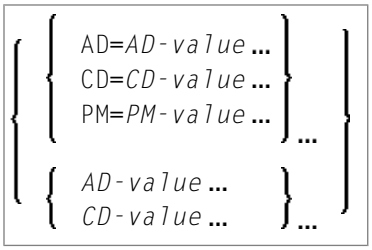
テキスト／属性割り当て

'text'	テキスト割り当て： 一重引用符で囲まれた文字列が表示されます。
'*c*'(*n*)	文字の繰り返し： フィールド値の直前に、一重引用符で囲まれた文字列が *n* 回表示されます。
attributes	フィールド表現と色属性： テキスト／フィールド表示にさまざまな属性を割り当てることができます。これらの属性と構文については、以下の「*出力属性*」で説明します。

	例： `WRITE TITLE 'TEXT' (BGR)` `WRITE TITLE 'TEXT' (B)` `WRITE TITLE 'TEXT' (BBLC)`

出力属性

attributes は、テキスト表示に使用される出力属性を示します。 可能な属性：



指定可能なセッションパラメータ値については、『パラメータリファレンス』ドキュメントの該当するセクションを参照してください。

- 「*AD - 属性定義*」の「*フィールド表現*」
- *CD - カラー定義*
- *PM - 出力モード*

Note: コンパイラは、実際には1つの出力フィールドに複数の属性値を受け入れます。 例えば、「AD=BDI」と指定できます。 ただし、この場合は最後の値のみが適用されます。 示した例では、値 "I" のみが有効になり、出力フィールドは強調表示されます。

例

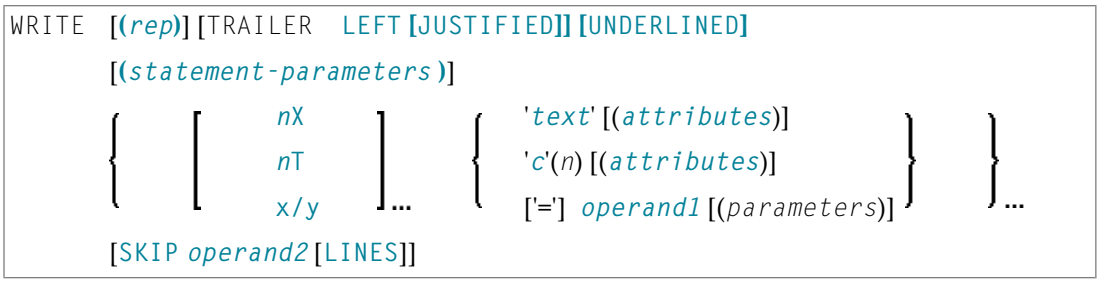
```
** Example 'WTIEX1': WRITE  (with TITLE option)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
END-DEFINE
*
*
```

```
FORMAT LS=70
*
WRITE TITLE LEFT JUSTIFIED UNDERLINED
      *TIME 3X 'PEOPLE LIVING IN NEW YORK CITY'
      11X 'PAGE:' *PAGE-NUMBER
SKIP 1
*
FIND EMPL-VIEW WITH CITY = 'NEW YORK'
  DISPLAY NAME FIRST-NAME 3X JOB-TITLE
END-FIND
END
```

プログラム **WTIEX1** の出力：

```
09:33:16.5   PEOPLE LIVING IN NEW YORK CITY           PAGE:      1
------------------------------------------------------------------------

        NAME              FIRST-NAME                CURRENT
                                                   POSITION
-------------------- --------------------   -------------------------

RUBIN                SYLVIA                 SECRETARY
WALLACE              MARY                   ANALYST
```

135 WRITE TRAILER



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**AT END OF PAGE** | **AT TOP OF PAGE** | **CLOSE PRINTER** | **DEFINE PRINTER** | **DISPLAY** | **EJECT** | **FORMAT** | **NEWPAGE** | **PRINT** | **SKIP** | **SUSPEND IDENTICAL SUPPRESS** | **WRITE** | **WRITE TITLE**

関連機能グループ：「*出力レポートの作成*」

機能

WRITE TRAILERステートメントは、ページの下にテキストまたは変数の内容を出力するために使用します。

『*プログラミングガイド*』の次のセクションも参照してください。

- *データ出力制御*
- *レポート指定 - (rep) 表記*
- *出力ページのレイアウト*
- *ページトレーラ - WRITE TRAILER ステートメント*
- *テキスト表記*

処理

このステートメントは非手続き型なので、プログラム内の位置ではなくイベントによって実行されます。

このステートメントは、エンドオブページ条件またはエンドオブデータ条件が検出されたとき、あるいはSKIPやNEWPAGEステートメントによって改ページが行われたときに実行されます。EJECTステートメントでは実行されません。

エンドオブページ条件は、DISPLAY/WRITEステートメントの処理が完了してから評価されます。DISPLAY/WRITEステートメントで複数行の出力を行う場合、エンドオブページ条件になる前に物理ページのオーバーフローを起こすことがあります。

レポートが、異なるオブジェクト内のステートメントによって作成される場合、WRITE TRAILER ステートメントは、END-OF-PAGE（ページ終了）処理を行うステートメントと同じオブジェクトで使用されている場合に限って実行されます。

論理ページサイズ

トレイラ情報を正しく同一ページの下に出力するには、論理ページサイズ（セッションパラメータ PS で指定）を物理ページサイズより小さく指定します。

制約

- WRITE TRAILER は 1 レポートにつき 1 回だけ指定できます。
- WRITE TRAILER は特殊条件ステートメントブロック内では指定できません。
- WRITE TRAILER はサブルーチン内では指定できません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット													ステートメント参照	ダイナミック定義
operand1		S	A	G	N	A	U	N	P	I	F	B	D	T	L		G	O	可	不可
operand2	C	S						N	P	I		B							可	不可

構文要素の説明：

(*rep*)	レポート指定： 複数のレポートを作成するときは、表記 (*rep*) を使用して、WRITE TRAILER ステートメントを適用するレポートの ID を指定できます。 レポート ID として、範囲 0～31 の値、または DEFINE PRINTER ステートメントを使用して割り当てた論理名を指定できます。 (*rep*) 指定がない場合、WRITE TRAILER ステートメントは最初のレポート（レポート 0）に適用されます。 Natural で作成される出力レポートの形式を制御する方法については、『*プログラミングガイド*』の「*データ出力制御*」を参照してください。
LEFT JUSTIFIED **UNDERLINED**	桁揃えおよび下線 デフォルトでは、トレーラ行は中央揃えで下線なしです。

	LEFT JUSTIFIED と UNDERLINED を指定して上書きできます。 UNDERLINEDを指定した場合、下線文字（デフォルトの文字またはセッションパラメータ UC で指定された文字）が、トレイラの下に行サイズ（セッションパラメータ LS）の幅だけ出力されます。 Naturalでは、すべての間隔およびタブ指定を適用して行を作成してから、行全体を中央揃えします。 例えば、最初の要素としての "10T" 表記により、中央揃えされたヘッダーは右側の 5 の位置に配置されます。
statement-parameters	ステートメントレベルでのパラメータ定義： 1つまたは複数のパラメータをカッコで囲んで、ステートメントレベルで、つまり WRITE TRAILER ステートメントの直後に指定できます。 この方法で指定した各パラメータは、以前に GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードでのみ）、または FORMAT ステートメントで指定したパラメータを上書きします。 複数のパラメータを指定する場合は、各エントリ間に1つ以上の空白を配置する必要があります。 エントリを 2 行のステートメント行に分割することはできません。 **注意:** ここで適用されるパラメータ設定は、変数フィールドにのみ関連し、テキスト定数には影響しません。 テキスト定数にフィールド属性を設定する場合は、この要素に属性を明示的に設定する必要があります（「*要素（フィールド）レベルでのパラメータ定義*」を参照）。 使用可能なパラメータについては、『ステートメント』ドキュメントの「WRITE」で「*パラメータのリスト*」を参照してください。
***n*X** ***n*T** *x*/*y*	フォーマット表記と要素間隔： 「*フォーマット表記と要素間隔*」（後述）を参照してください。
'*text*' '*c*'(*n*) *attributes*	テキスト／属性割り当て： 「*テキスト／属性割り当て*」（後述）を参照してください。
operand1	トレイラ情報： *operand1* は、トレイラ情報として出力するフィールド（複数可）を表します。
parameters	要素（フィールド）レベルでのパラメータ定義： 1 つまたは複数のパラメータをカッコで囲んで、要素（フィールド）レベルで、つまり *operand1* の直後に指定できます。この方法で指定した各パラメータは、以前にステートメントレベル、GLOBALS コマンド、SET GLOBALS ステートメント（レポーティングモードでのみ）、または FORMAT ステートメントで指定した対応するパラメータを上書きします。

	複数のパラメータを指定する場合は、各エントリ間に1つ以上の空白を配置する必要があります。 エントリを 2 行のステートメント行に分割することはできません。 使用可能なパラメータについては、『ステートメント』ドキュメントの「WRITE」で「*パラメータのリスト*」を参照してください。
SKIP *operand2* **LINES**	省略される行： SKIP を使用して、トレイラ行の下に何行か空けることができます。空ける行数（*operand2*）は数値定数または数値変数の内容で指定します。 **注意:** WRITE TRAILER の後の SKIP は独立したステートメントでなく、常に WRITE TRAILER ステートメントの SKIP 節として解釈されます。 WRITE TRAILER ステートメントの後に独立した SKIP ステートメントが必要な場合は、セミコロン（;）を使用して 2 つのステートメントを互いに区別します。

フォーマット表記と要素間隔

*n***X**	列の間隔： この表記により、列の間に *n* 個のスペースが挿入されます。
*n***T**	タブ設定： *n*T 表記により、位置 *n* を出力するように位置指定（タブ設定）されます。 後方への位置指定はできません。
x/*y*	*x*/*y* 位置指定： 次の要素は最後のステートメント出力の *x* 行下、列 *y* の先頭に配置されます。 *y* を "0" にすることはできません。 同じ行で後方に位置指定することはできません。

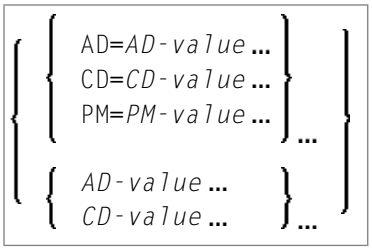
テキスト／属性割り当て

'text'	テキスト割り当て： 一重引用符で囲まれた文字列が表示されます。
'*c*'(*n*)	文字の繰り返し： フィールド値の直前に、一重引用符で囲まれた文字列が *n* 回表示されます。
attributes	フィールド表現と色属性： テキスト／フィールド表示にさまざまな属性を割り当てることができます。これらの属性と構文については、以下の「*出力属性*」で説明します。

	例： `WRITE TRAILER 'TEXT' (BGR)` `WRITE TRAILER 'TEXT' (B)` `WRITE TRAILER 'TEXT' (BBLC)`

出力属性

attributes は、テキスト表示に使用される出力属性を示します。 可能な属性：



指定可能なセッションパラメータ値については、『パラメータリファレンス』ドキュメントの該当するセクションを参照してください。

- 「*AD - 属性定義*」の「*フィールド表現*」
- *CD - カラー定義*
- *PM - 出力モード*

Note: コンパイラは、実際には1つの出力フィールドに複数の属性値を受け入れます。 例えば、「AD=BDI」と指定できます。 ただし、この場合は最後の値のみが適用されます。 示した例では、値 "I" のみが有効になり、出力フィールドは強調表示されます。

例

```
** Example 'WTLEX1': WRITE (with TRAILER option)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
END-DEFINE
*
FORMAT PS=15
```

```
WRITE TITLE LEFT JUSTIFIED UNDERLINED
      *TIME 3X 'PEOPLE LIVING IN BARCELONA'
      14X 'PAGE:' *PAGE-NUMBER
SKIP 1
*
WRITE TRAILER LEFT JUSTIFIED UNDERLINED
      / 'CITY OF BARCELONA REGISTER'
*
LIMIT 10
FIND EMPL-VIEW WITH CITY = 'BARCELONA'
  DISPLAY NAME FIRST-NAME 3X JOB-TITLE
END-FIND
END
```

プログラム **WTLEX1** の出力 **-** ページ **1**：

```
09:36:09.5   PEOPLE LIVING IN BARCELONA              PAGE:      1
------------------------------------------------------------------------------

        NAME              FIRST-NAME                CURRENT
                                                   POSITION
-------------------- --------------------   -------------------------

DEL CASTILLO         ANGEL                  EJECUTIVO DE VENTAS
GARCIA               M. DE LAS MERCEDES     SECRETARIA
GARCIA               ENDIKA                 DIRECTOR TECNICO
MARTIN               ASUNCION               SECRETARIA
MARTINEZ             TERESA                 SECRETARIA
YNCLAN               FELIPE                 ADMINISTRADOR
FERNANDEZ            ELOY                   OFICINISTA
TORRES               ANTONI                 OBRERA

CITY OF BARCELONA REGISTER
------------------------------------------------------------------------------
```

プログラム **WTLEX1** の出力 **-** ページ **2**：

```
09:37:26.0   PEOPLE LIVING IN BARCELONA              PAGE:      2
------------------------------------------------------------------------------

        NAME              FIRST-NAME                CURRENT
                                                   POSITION
-------------------- --------------------   -------------------------

RODRIGUEZ            VICTORIA               SECRETARIA
GARCIA               GERARDO                INGENIERO DE PRODUCCION

CITY OF BARCELONA REGISTER
------------------------------------------------------------------------------
```

136 WRITE WORK FILE

```
WRITE WORK [FILE] work-file-number [VARIABLE] operand1 ...
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

関連ステートメント：**DEFINE WORK FILE** | **READ WORK FILE** | **CLOSE WORK FILE**

関連機能グループ：「*ワークファイル／PC ファイルの制御*」

機能

`WRITE WORK FILE` ステートメントは、物理順ワークファイルにレコードを書き込むために使用します。

このステートメントはバッチモードでのみ使用できます。

1つのプログラム内または処理ループ内でワークファイルを作成し、そのファイルを後続の別の処理ループまたはプログラムで `READ WORK FILE` ステートメントを使用して読み取ることもできます。

Note: Unicode およびコードページのサポートについては、『*Unicode およびコードページのサポート*』ドキュメントの「*Windows、UNIX、および OpenVMS プラットフォーム上のワークファイルと出力ファイル*」を参照してください。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand1	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	可	不可

Note: ワークファイルタイプ ENTIRECONNECTION または TRANSFER を使用しているとき、*operand1* のフォーマットを C または G にすることはできません。

構文要素の説明：

work-file-number	ワークファイル番号： 使用するワークファイル番号（Natural に定義されたもの）を指定します。
VARIABLE	可変エントリ： 同一ワークファイルに対し、異なる WRITE WORK FILE ステートメントで、異なるフィールドを持つレコードを書き込むことができます。この場合、すべての WRITE WORK FILE ステートメントに VARIABLE を指定する必要があります。 外部ファイルに書き込まれるレコードは可変フォーマットです。 オペランドリストにダイナミック変数（WRITE WORK FILE ステートメントの異なる実行に対してサイズが変わる）が含まれている場合、VARIABLE エントリはすべての WRITE WORK FILE ステートメントに指定する必要があります。 変数インデックス範囲： ワークファイルに配列を書き込む場合、配列に変数インデックス範囲を指定できます。 次に例を示します。 `WRITE WORK FILE work-file-number VARIABLE #ARRAY (I:J)`
operand1	フィールド： *operand1* では、ワークファイルに書き込むフィールドを指定します。ワークファイルに書き込むフィールドには、データベースフィールド、ユーザー定義変数、または READ WORK FILE ステートメントを使用して別のワークファイルから読み取ったフィールドを指定できます。 データベースの配列は、ワークファイルに書き込むオカレンスを示すインデックスの単一範囲で参照できます。データベースファイルのグループは、グループ名で参照できます。グループに属するすべてのデータベースフィールドがワークファイルに書き込まれます。

フィールドの外部表示

WRITE WORK FILE ステートメントで書き込まれるフィールドは、その内部定義に従って外部ファイルに表示されます。 フィールド値は編集されません。

A および B フォーマットのフィールドについては、外部ファイルでのバイト数は、Natural プログラムで定義された内部長さ定義と同じです。 値は編集されず、小数点は表示されません。

N フォーマットのフィールドについては、外部ファイルでのバイト数は、小数点の前後の内部桁数の合計です。 小数点は外部ファイルには表示されません。

Pフォーマットのフィールドについては、外部ファイルでのバイト数は、小数点の前後の桁数の合計に 1（符号用）を加算し、それを 2 で割ってバイト単位に切り上げたものです。

Note: フォーマット変換は、ワークファイルに書き込まれるフィールドに対しては行われません。

フィールド表現の例：

フィールド定義	出力レコード
#FIELD1 (A10)	10 バイト
#FIELD2 (B15)	15 バイト
#FIELD3 (N1.3)	4 バイト
#FIELD4 (N0.7)	7 バイト
#FIELD5 (P1.2)	2 バイト
#FIELD6 (P6.0)	4 バイト

Note: 数値フィールド（フォーマット N または P）に対する Natural システム関数 AVER、NAVER、SUM、または TOTAL をワークファイルに書き込むとき、これらのフィールドの内部長は 1 桁拡張されます（例：フォーマット P3 のフィールドの SUM は P4 に拡張されます）。 これにより、ワークファイルを読み取るときに注意する必要があります。

ラージおよびダイナミック変数の処理

ワークファイルタイプ	処理
ASCII **ASCII-COMPRESSED** **SAG**（バイナリ）	ワークファイルタイプ ASCII、ASCII-COMPRESSED、および SAG（バイナリ）では、ダイナミック変数は処理できず、エラーが生成されます。ただし、最大フィールド／レコード長が 32766 バイトのラージ変数は処理できます。
ENTIRECONNECTION	ワークファイルタイプ ENTIRE CONNECTION では、ダイナミック変数は処理できません。 ただし、このタイプでは、最大フィールド／レコード長が 1073741824 バイトのラージ変数を処理できます。
PORTABLE **UNFORMATTED**	2 つのワークファイルタイプ PORTABLE と UNFORMATTED を使用して、ラージおよびダイナミック変数をワークファイルに書き込んだりワークファイルから読み取ったりできます。これらのタイプには、ダイナミック変数に対するサイズ制限がありません。ただし、ラージ変数は最大フィールド／レコード長の 32766 バイトを超えることはできません。 ワークファイルタイプ PORTABLE の場合は、フィールド情報がワークファイル内に保存されます。 レコード内のフィールドサイズが現在のサイズと異なる場合は、ダイナミック変数は READ 中にサイズ変更されます。 WRITE WORK FILE ステートメントでは、フィールドは、そのバイト長で指定されたファイルに書き込まれます。 すべてのデータタイプ（DYNAMIC であ

ワークファイルタイプ	処理
	るかどうかに関わらず）は同じように扱われます。 構造情報は挿入されません。 Natural ではバッファリングメカニズムを使用するので、データが完全に書き込まれるのは CLOSE WORK の後のみであることが予測できます。 これは、Natural の稼働中にファイルが別のユーティリティで処理される場合に特に重要です。 READ WORK FILE ステートメントでは、固定長のフィールドは、その全体の長さで読み込まれます。 ファイルの終わりに到達すると、現在のフィールドの残りは空白で埋められます。 次のフィールドは変更されません。 データタイプが DYNAMIC の場合、ファイルが 1073741824 バイトを超えない限り、ファイルの残りの部分がすべて読み込まれます。 ファイルの終わりに到達すると、残りのフィールド（変数）は変更されないまま維持されます（通常の Natural の動作）。
CSV	ダイナミックおよびラージ変数の最大フィールド／レコード長は 32766 バイトです。 ダイナミック変数がサポートされます。 X-array は許可されていないので、エラーメッセージが表示されます。

例

```
** Example 'WWFEX1': WRITE WORK FILE
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'LONDON'
  WRITE WORK FILE 1
        PERSONNEL-ID NAME
END-FIND
*
END
```

137 SQL ステートメント

Natural DML ステートメントの他に、Natural プログラムで SQL ステートメントを使用して直接 SQL を扱うことができます。

次の SQL ステートメントが有効です。

CALLDBPROC | COMMIT | DELETE | INSERT | PROCESS SQL | READ RESULT SET | ROLLBACK | SELECT | UPDATE

Note: Natural アプリケーションの移植性に関して、Natural がサポートする全データベースシステムに対して使用できる FIND、READ のような Natural DML ステートメントとは別に、Natural SQL ステートメントは SQL 対応のデータベースシステムにだけ使用できることに注意してください。

このセクションでは、次のトピックについて説明します。

- 一般セットと拡張セット
- 基本構文項目
- **Natural** ビューの概念
- スカラー式
- 検索条件
- 選択式
- フレキシブル **SQL**
- SQL ステートメント - アルファベット順

「フレキシブル SQL」 と呼ばれる SQL ステートメント発行の追加機能では、任意の SQL 構文を使用できます。

138 一般セットと拡張セット

Naturalプログラミング言語内で使用できるSQLステートメントは、次の2つの異なる構文セットから成ります。

- 一般セット
  基本的に標準のSQL構文定義に対応しており、Naturalがサポートする SQL 対応の各データベースシステム用に提供されています。 一般セットはすべてのSQLデータベースに対して有効です。
- 拡張セット
  サポートされているデータベースシステム特有の機能をサポートするため、一般セットに特別な機能拡張を提供しています。 現在、拡張セットは部分的に使用でき、DB2 データベースに対してのみ有効です。

Natural SQLステートメントのドキュメントでは、主にNatural SQL一般セットについて説明しています。 ステートメント構文は、できる限りSQLの文献に記載されている構文に従っているため、これらの文献も参照してください。

139 基本構文項目

このchapterでは、個々のSQLステートメントの説明には詳しく記載されていない基本構文項目について説明します。 これらの項目は次のとおりです。

このchapterでは、次のトピックについて説明します。

定数

Natural SQL ステートメントの構文記述で使用される定数は次のとおりです。

- *constant*
- *integer*

これらの各項目については、以下で説明します。

constant	項目 *constant* は、常に Natural 定数を表します。
integer	項目 *integer* は、常に整数の定数を表します。

Note: 小数点表記の文字（セッションパラメータ DC）がコンマ（,）に設定されている場合、指定した数値定数の直後にコンマを続けてはならず、定数とコンマを空白文字で区切る必要があります。そうしないと、エラーまたは不正な結果が生じます。

無効な構文：	有効な構文：
`VALUES (1,'A')  leads to a syntax error` `VALUES (1,2,3)  leads to wrong results`	`VALUES (1 ,'A')` `VALUES (1 ,2 ,3)`

名前

Natural SQL ステートメントの構文記述で使用される名前は次のとおりです。

- *authorization-identifier*
- *ddm-name*
- *view-name*
- *column-name*
- *table-name*
- *correlation-name*

これらの各項目については、以下で説明します。

authorization-identifier	項目 *authorization-identifier*（作成者名とも呼ばれる）は、データベーステーブルとビューを修飾するために使用します。 下記も参照してください。
ddm-name	項目 *ddm-name* は、常に Natural ユーティリティ SYSDDM で作成された Natural DDM の名前を表します。
view-name	項目 *view-name* は、常に DEFINE DATA ステートメントで定義された Natural ビューの名前を表します。
column-name	項目 *column-name* は、常に物理データベース列の名前を表します。
table-name	構文： *authorization-identifier ddm-name* このセクションの項目 *table-name* は、SQL 基本テーブルと SQL ビューテーブルの両方を参照するために使用します。 Natural DDM は、使用するテーブル用に作成されている必要があります。 このような DDM の名前は、対応するデータベーステーブル名またはビュー名と同じにする必要があります。 *authorization-identifier* データベーステーブルまたはビューの *authorization-identifier* は、2 とおりの方法で指定できます。 1 つの方法は標準 SQL 構文に対応しています。この場合、*authorization-identifier* とテーブル名はピリオドで区切ります。この形式を使用する場合、*authorization-identifier* を使用せずに、DDM の名前をデータベーステーブルの名前と同じにする必要があります。 例： `DEFINE DATA LOCAL` `01 PERS VIEW OF PERSONNEL` `  02 NAME` `  02 AGE` `END-DEFINE` `SELECT *` `  INTO VIEW PERS` `  FROM SQL.PERSONNEL` `...` 別の方法として、*authorization-identifier* を DDM 名の一部として定義できます。 DDM 名は、*authorization-identifier* とデータベーステーブル名をハイフン（-）で区切って構成されます。

	authorization-identifier とテーブル名の間のハイフンは内部的にピリオドに変換されます。 **注意:** この形式の DDM 名は、FIND または READ ステートメントでも使用できます。これらのステートメントに適用される DDM 命名規則に準拠しているためです。 例： `DEFINE DATA LOCAL` `01 PERS VIEW OF SQL-PERSONNEL` `  02 NAME` `  02 AGE` `END-DEFINE` `SELECT *` `  INTO VIEW PERS` `  FROM SQL-PERSONNEL` `...` *authorization-identifier* が明示的にも DDM 名内にも指定されていない場合、SQL データベースシステムによって決定されます。 *table-names* は、SELECT ステートメントで使用されるほか、DELETE、INSERT、および UPDATE ステートメントにも指定できます。 例： `...` `DELETE FROM SQL.PERSONNEL` `  WHERE AGE IS NULL` `  ...` `...` `INSERT INTO SQL.PERSONNEL (NAME,AGE)` `  VALUES ('ADKINSON',35)` `...` `...` `UPDATE SQL.PERSONNEL` `SET SALARY = SALARY * 1.1` `WHERE AGE > 30` `...`
correlation-name	項目 *correlation-name* は、テーブル名のエイリアス名を表します。列名を修飾するために使用できます。また、SELECT ステートメントの INTO 節で使用すると、Natural ビューでフィールドを暗黙的に修飾できます。

	例： `DEFINE DATA LOCAL` `01 PERS-NAME    (A20)` `01 EMPL-NAME    (A20)` `01 AGE          (I2)` `END-DEFINE` `...` `SELECT X.NAME , Y.NAME , X.AGE` `  INTO PERS-NAME , EMPL-NAME , AGE` `  FROM SQL-PERSONNEL X , SQL-EMPLOYEES Y` `  WHERE X.AGE = Y.AGE` `END-SELECT` `...` ほとんどの場合、*correlation-names*を使用する必要はありませんが、使用するとステートメントがわかりやすくなります。

パラメータ

parameter

[:] *host-variable* [INDICATOR **[:]** *host-variable*] [LINDICATOR **[:]** *host-variable*]

構文の各項目については、以下で説明します。

host-variable	*host-variable*は、SQL ステートメントで参照される Natural ユーザー定義変数（システム変数ではない）です。 個々のフィールドまたは Natural ビューの一部として定義できます。 受信側フィールドとして（INTO 節などで）定義する場合、*host-variable* では値がデータベースシステムによって割り当てられている変数を識別します。 送信側フィールドとして（WHERE 節などで）定義する場合、*host-variable*ではプログラムからデータベースシステムに渡される値を指定します。 「*Natural フォーマットと SQL データタイプ*」も参照してください。
[:]	コロン： SQL標準に準拠するために、ホスト変数の先頭にコロン（:）を付けることもできます。フレキシブル SQL で使用する場合、ホスト変数はコロンで修飾する必要があります。

	例： `SELECT NAME INTO :#NAME FROM PERSONNEL` `WHERE AGE = :VALUE` 変数名が SQL 予約語と同一の場合、コロンは常に必要です。 ホスト変数または列の参照が可能な状況では、コロンなしの名前を使用すると、列への参照として解釈されます。
INDICATOR	**INDICATOR** 節： INDICATOR 節は、「空値」（つまり、値なし）と実際の値 "0" または 「空白」 を区別するためのオプション機能です。 受信側 *host-variable*（ターゲットフィールド）で指定する場合、INDICATOR *host-variable*（空値インジケータフィールド）によって、取得される列が「空値」であるかどうかを調べることができます。 例： `DEFINE DATA LOCAL` `1 NAME      (A20)` `1 NAMEIND   (I2)` `END-DEFINE` `SELECT *` `  INTO NAME INDICATOR NAMEIND` `...` この例では、NAME は受信側 *host-variable* を表し、NAMEIND は空値インジケータフィールドを表します。 空値インジケータフィールドが指定されており、取得される列が空値の場合、空値インジケータフィールドの値は負になり、そのデータタイプに応じてターゲットフィールドが "0" または 「空値」 に設定されます。 それ以外の場合、空値インジケータフィールドの値は "0" 以上になります。 送信側 *host-variable*（ソースフィールド）で指定する場合、空値インジケータフィールドはこのフィールドに空値を指定するために使用されます。

	例： `DEFINE DATA LOCAL` `1 NAME      (A20)` `1 NAMEIND   (I2)` `UPDATE ...` `SET NAME = :NAME INDICATOR :NAMEIND` `WHERE ...` この例では、:NAMEは送信側ホスト変数を表し、:NAMEINDは空値インジケータフィールドを表します。 空値インジケータフィールドの入力として負の値を入力すると、空値がデータベース列に割り当てられます。 INDICATOR *host-variable*のフォーマット／長さはI2です。
LINDICATOR	**LINDICATOR**節： LINDICATOR節は、可変長（例えば、VARCHARまたはLONG VARCHARタイプ）の列をサポートするためのオプション機能です。 受信側*host-variable*（ターゲットフィールド）で指定する場合、LINDICATOR *host-variable*（長さインジケータフィールド）にはデータベースによってターゲットフィールドに実際に返された文字数が入ります。 ターゲットフィールドは常に空白で埋められます。 VARCHARまたはLONG VARCHAR列にターゲットフィールドに収まりきらない文字が含まれている場合、長さインジケータフィールドは実際に返された長さ（つまり、ターゲットフィールドの長さ）に設定され、空値インジケータフィールド（指定されている場合）はこの列の合計長に設定されます。 例 `DEFINE DATA LOCAL` `1 ADDRESSLIND (I2)` `1 ADDRESS     (A50/1:6)` `END-DEFINE` `SELECT *` `  INTO :ADDRESS(*) LINDICATOR :ADDRESSLIND` `  ...` この例では、:ADDRESS(*)はアドレス指定されたVARCHARまたはLONG VARCHAR列の最初の300バイト（利用可能な場合）を受け取るターゲットフィールドを表し、:ADDRESSLINDは実際に返された文字数を含む長さインジケータフィールドを表します。 送信側*host-variable*（ソースフィールド）で指定する場合、長さインジケータフィールドではデータベースに渡されるソースフィールドの文字数を指定します。

例：

```
DEFINE DATA LOCAL
1 NAMELIND  (I2)
1 NAME      (A20)
1 AGE       (I2)
END-DEFINE
MOVE  4     TO NAMELIND
MOVE 'ABC%' TO NAME
SELECT AGE
  INTO :AGE
WHERE NAME LIKE :NAME LINDICATOR :NAMELIND
  ...
```

LINDICATOR *host-variable* のフォーマット／長さは I2 または I4 です。パフォーマンス上の理由から、対応するターゲットフィールドまたはソースフィールドの直前に指定する必要があります。そうしないと、フィールドはランタイムに一時ストレージにコピーされます。

LINDICATOR フィールドを I2 フィールドとして定義すると、対応する列の送受信には SQL データタイプ VARCHAR が使用されます。 LINDICATOR *host-variable* を I4 として指定すると、ラージオブジェクトデータタイプ（CLOB/BLOB）が使用されます。

フィールドを DYNAMIC として定義すると、列は内部ループでは実際の長さまで読み取られます。 LINDICATOR フィールドと *LENGTH はこの長さに設定されます。 固定長フィールドの場合、列は定義された長さまで読み込まれます。いずれの場合も、フィールドは LINDICATOR フィールドで定義した値まで書き込まれます。

固定長フィールドの定義には、I2 として指定した LINDICATOR フィールドを使用してください。 VARCHAR 列にこの固定長フィールドに収まりきらない文字が含まれている場合、長さインジケータフィールドは実際に返された長さに設定され、空値インジケータフィールド（指定されている場合）はこの列の合計長に設定されます（取得）。これは32KB以上の固定長フィールドには適用されません（長さが空値インジケータフィールドに収まらないため）。

Natural フォーマットと SQL データタイプ

ホスト変数の Natural フォーマットは、次の表に従って SQL データタイプに変換されます。

Natural フォーマット／長さ	SQL データタイプ
A*n*	CHAR (*n*)
B2	SMALLINT
B4	INT
B*n*（*n* は 2 または 4 以外）	CHAR (*n*)
F4	REAL
F8	DOUBLE PRECISION
I2	SMALLINT
I4	INT
N*nn.m*	NUMERIC (*nn*+*m*,*m*)
P*nn.m*	NUMERIC (*nn*+*m*,*m*)
T	TIME
D	DATE
G*n*（ビューフィールドのみ）	GRAPHIC (*n*)

Natural では、変換後の SQL データタイプがデータベース列と互換性があるかどうかはチェックされません。 フォーマット N のフィールドを除き、データ変換は行われません。

また、Natural SQL では、標準 Natural フォーマットに対する次の拡張を使用できます。

- フォーマット A の 1 次元配列を使用して、253 バイトを超える英数字列をサポートできます。この配列はインデックス 1 から定義する必要があり、インデックスとしてアスタリスク（*）を使用することでのみ参照できます。 対応する SQL データタイプは `CHAR (n)` です。ここで、*n* は配列の合計バイト数です。
- キーワード `LINDICATOR` で指定した特別なホスト変数を使用して、可変長列をサポートできます。 対応する SQL データタイプは `VARCHAR (n)` です。`LINDICATOR` 節も参照してください。
- Entire Access では、Natural フォーマットの日付（D）および時刻（T）を使用できます。これらのフォーマットは、対応するデータベース依存フォーマットに変換されます（詳細については、Entire Access ドキュメントを参照）。

`LINDICATOR` フィールドを使用せずに 1 次元配列として指定した送信側フィールドは、SQL データタイプ `VARCHAR` に変換されます。 長さは配列の合計バイト数になり、末尾の空白は考慮されません。

140 Natural ビューの概念

いくつかの Natural SQL ステートメントも Natural ビューの使用をサポートします。

Naturalビューをパラメータリストの代わりに指定できます。ビューの各フィールド（グループフィールド、再定義フィールド、および "L@" や "N@" の接頭文字付きフィールド以外）は 1 パラメータ（ホスト変数）に対応します。

名前に接頭文字 "L@" や "N@" が付加されたフィールドは、対応するマスタフィールド（同じ名前のフィールド）とともにのみ存在させることができます。

- L@ フィールドは LINDICATOR フィールドに変換されます。
- N@ フィールドは INDICATOR フィールドに変換されます。

ビュー定義で、L@ フィールドは、適用先のマスタフィールドの直前に指定してください。

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 PERSID     (I4)
  02 NAME       (A20)
  02 N@NAME    (I2)                    /* null indicator of NAME
  02 L@ADDRESS (I2)                    /* length indicator of ADDRESS
  02 ADDRESS    (A50/1:6)
  02 N@ADDRESS (I2)                    /* null indicator of ADDRESS
01 #PERSID      (I4)
END-DEFINE
   ...
SELECT *
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE PERSID = #PERSID
```

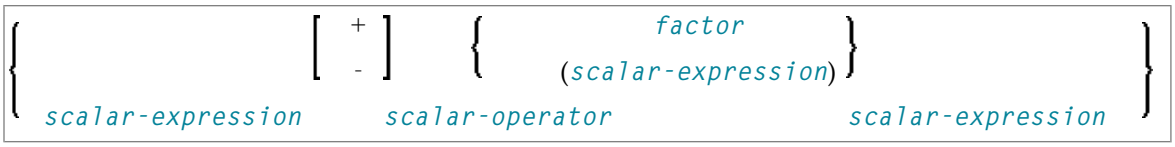
```
   ...
END-SELECT
```

上記の例は下記の例と同等です。

```
   ...
SELECT *
  INTO PERSID,
       NAME INDICATOR N@NAME,
       ADDRESS(*)INDICATOR N@ADDRESS LINDICATOR L@ADDRESS
  FROM SQL-PERSONNEL
  WHERE PERSID = #PERSID
   ...
END-SELECT
```

Note: Natural for Windows、Natural for UNIX、または Natural for OpenVMS で varchar データタイプにアクセスする際には、対応する長さインジケータ変数がビュー内に存在している必要があります。

141 スカラー式



このchapterでは、次のトピックについて説明します。

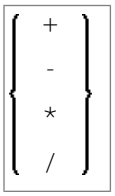
スカラー式

*scalar-expression*は、ファクタまたはスカラー演算子を含む他のスカラー式で構成されます。

参照の優先順位に関して、スカラー式は次のように処理されます。

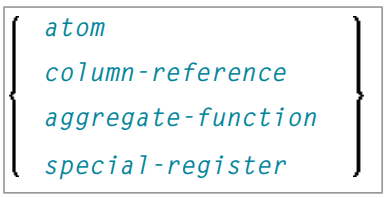
- 非修飾変数名がスカラー式に指定されるとき、最初に変数名は参照テーブルの列名として解決されます。
- 指定された名前の列が参照テーブルで有効でない場合、Natural はこの変数を Natural ユーザー定義変数（ホスト変数）として解決しようとします。

スカラー演算子



scalar-operator とは上記のいずれかの演算子です。演算子 "-" および "/" を使用する場合は、先行する演算子との間を最低 1 つの空白で区切る必要があります。

ファクタ



factor は、上図のいずれかの項目で構成されます。これらの項目について次に説明します。

原子

atom は、*parameter* または *constant* です。「*基本構文項目*」も参照してください。

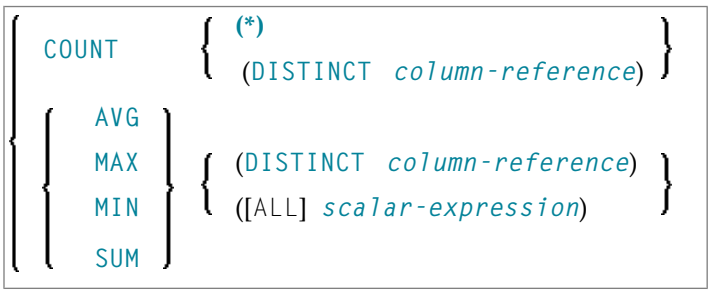
列参照

column-reference は、オプションで *table-name* または *correlation-name*（「*基本構文項目*」も参照）で修飾された列名です。 修飾された名前は修飾されない名前よりも明確であり、ときには修飾が必要となります。

Note: このような場合には、テーブル名を認可 ID で明示的に修飾しないでください。テーブル名を修飾するときは、代わりに相関名を使用してください。

table-name または *correlation-name* で列を参照する場合、列は対応するテーブル内に含まれている必要があります。 *table-name* も *correlation-name* も指定されていない場合、各列は FROM 節に指定されたテーブルの 1 つに存在している必要があります。

集積関数



SQL は基本的な取得機能拡張のために多くの特別関数を提供しています。 現在有効で Natural がサポートする、いわゆる SQL 集積関数は次のとおりです。

AVG	列の平均値
COUNT	列の値の個数
MAX	列の最大値
MIN	列の最小値
SUM	列の合計値

COUNT(*) は別にして、上記の関数は引数（つまり、単一の列または *scalar-expression*）のスカラー値の集合を処理します。その結果としてスカラー値を生成します。

例：

```
DEFINE DATA LOCAL
1 AVGAGE   (I2)
END-DEFINE
...
SELECT AVG (AGE)
  INTO AVGAGE
  FROM SQL-PERSONNEL
  ...
```

関数が適用される前に余分な重複値を削除するため、一般的に、引数の前にキーワード DISTINCT をオプションで指定できます。

DISTINCT を指定する場合、引数は単一の列名にする必要があります。DISTINCT を指定しない場合、引数は一般的な *scalar-expression* の構成にすることもできます。

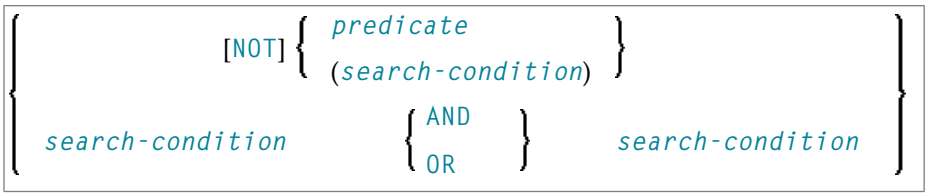
重複値を削除せずにテーブルのすべての行をカウントする特別関数 COUNT(*) に DISTINCT を指定することはできません。

特別レジスタ

```
USER
```

special-register に対する参照はスカラー値を返します。

142 検索条件



このchapterでは、次のトピックについて説明します。

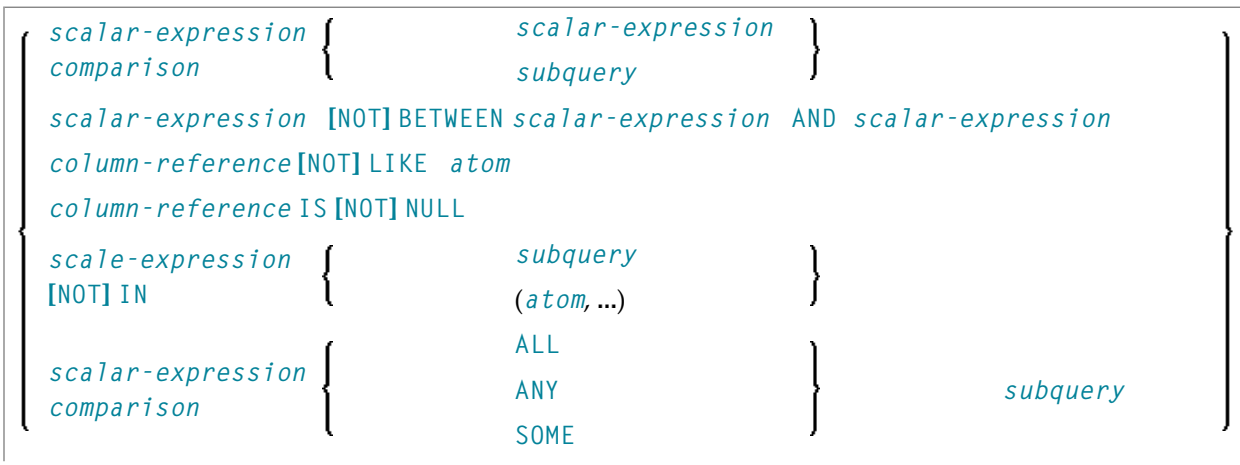
検索条件

*search-condition*は、単純な*predicate*、またはブール演算子AND、OR、NOT、およびカッコ（評価の順序を示す必要がある場合）を組み合せた複数の*search-conditions*で構成できます。

例

```
DEFINE DATA LOCAL
01 NAME   (A20)
01 AGE    (I2)
END-DEFINE
...
SELECT *
  INTO NAME, AGE
  FROM SQL-PERSONNEL
  WHERE AGE = 32 AND NAME > 'K'
END-SELECT
...
```

属性



```
  EXISTS subquery
```

*predicate*には「true」（真）、「false」（偽）、「unknown」（不明）の条件を指定します。

*search-condition*の*predicate*は、単純な比較演算または複雑な比較演算、あるいは他の種類の条件で構成できます。

例：

```
SELECT NAME, AGE
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE BETWEEN 20 AND 30
    OR AGE IN ( 32, 34, 36 )
    AND NAME LIKE '%er'
    ...
```

Note: パーセント記号（%）はNatural端末コマンドと混同するおそれがあります。 その場合は、"%" と異なる端末コマンド制御文字を定義する必要があります。

各属性については、以降のトピックで説明します（属性の詳細については、関連文献を参照してください）。 上記の構文に関する属性は次のように呼ばれます。

- 比較属性
- BETWEEN 属性
- LIKE 属性
- NULL 属性
- IN 属性
- 数量属性
- EXISTS 属性

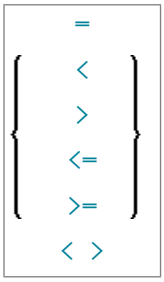
比較属性

```
scalar-expression comparison { scalar-expression | subquery }
```

比較属性は2つの値を比較します。

詳細については、*scalar-expression*を参照してください。

比較



comparison は次のいずれかの演算子です。

=	等しい
<	より小さい
>	より大きい
<=	より小さいまたは等しい
>=	より大きいまたは等しい
< >	等しくない

サブクエリ

subquery は同様の他の式内でネストされた *select-expression* です。

例：

```
DEFINE DATA LOCAL
1 #NAME    (A20)
1 #PERSNR  (I4)
END-DEFINE
...
SELECT NAME, PERSNR
  INTO #NAME, #PERSNR
  FROM SQL-PERSONNEL
  WHERE PERSNR IN
    ( SELECT PERSNR
        FROM SQL-AUTOMOBILES
        WHERE COLOR = 'black' )
        ...
END-SELECT
```

詳細については、「*選択式*」を参照してください。

BETWEEN 属性

```
scalar-expression [NOT] BETWEEN scalar-expression AND scalar-expression
```

BETWEEN 属性は範囲値で値を比較します。

詳細については、*scalar-expression* を参照してください。

LIKE 属性

```
column-reference [NOT] LIKE atom
```

LIKE 属性は特定のパターンを持つ文字列を検索します。

詳細については、*column-reference* および *atom* を参照してください。

NULL 属性

```
column-reference IS [NOT] NULL
```

NULL 属性は空値のテストを行います。

詳細については、*column-reference* を参照してください。

IN 属性

```
scalar-expression [NOT] IN { subquery ... }
                           { (atom)       }
```

IN 属性は値の集合で値を比較します。

詳細については、*scalar-expression* および *atom* を参照してください。

詳細については、*subquery* を参照してください。

数量属性

```
scalar-expression comparison { ALL | ANY | SOME } subquery
```

数量属性は値の集合で値を比較します。

詳細については、*scalar-expression*、*comparison*、および *subquery* を参照してください。

EXISTS 属性

```
EXISTS subquery
```

EXISTS 属性は特定の行の存在についてテストを行います。

EXISTS 属性は、*subquery* の評価結果が空でない場合、つまり、*subquery* の WHERE 節の検索条件を満たす *subquery* の FROM テーブルに最低 1 つのレコード（行）が存在する場合にのみ、真であると評価します。

EXISTS の例：

```
DEFINE DATA LOCAL
1 #NAME    (A20)
END-DEFINE
...
SELECT NAME
  INTO #NAME
  FROM SQL-PERSONNEL
  WHERE EXISTS
    ( SELECT *
      FROM SQL-EMPLOYEES
      WHERE PERSNR > 1000
        AND NAME < 'L' )
        ...
END-SELECT
...
```

詳細については、*subquery* を参照してください。

143 選択式

```
SELECT selection table-expression
```

select-expression では、結果テーブルを指定します。 次のステートメントで使用されます。
INSERT | **SELECT**

このchapterでは、次のトピックについて説明します。

選択

```
[ ALL      ] { {scalar-expression [[AS] correlation-name]}, ... }
[ DISTINCT ] { *                                                 }
```

selection では、選択する項目を指定します。

ALL/DISTINCT

select-expression の結果から重複行が自動的に取り除かれることはありません。 取り除く処理を要求するには、キーワード DISTINCT を指定します。

DISTINCT に代わるのが ALL です。 指定がない場合は ALL とみなされます。

スカラー式

単純な列名の代わりに、または列名とともに、スカラー演算子および計算値を与えるスカラー関数を含む一般的な *scalar-expressions* を含めることもできます（「スカラー式」も参照）。

例：

```
SELECT NAME, 65 - AGE
  FROM SQL-PERSONNEL
  ...
```

相関名

結果の列に対するエイリアス名として、*scalar-expression* に *correlation-name* を割り当てることができます。

correlation-name は一意にする必要はありません。 結果の列に *correlation-name* を指定しない場合、対応する *column-name* が使用されます（結果の列が列名から派生する場合。派生しない場合は結果テーブルは名前を持ちません）。 結果の列の名前は、SELECT ステートメントの ORDER BY 節の列名などに使用できます。

アスタリスク表記 -*

FROM 節に指定された全テーブルのすべての列が選択されます。

例：

```
SELECT *
  FROM SQL-PERSONNEL, SQL-AUTOMOBILES
  ...
```

テーブル式

```
FROM table-reference,...
 [WHERE search-condition]
 [GROUP BY column-reference,...]
 [HAVING search-condition]
```

table-expression では、どこから、どのような条件に従って行を選択するかを指定します。

FROM 節

```
FROM table-reference,...
```

テーブル参照

```
table-name [[AS] correlation-name]
subquery [AS] correlation-name
joined-table
```

FROM 節に指定したテーブルには、選択リストに使用した列が含まれている必要があります。

単一テーブルを指定するか、サブクエリまたは「ジョイン」処理（下記参照）による中間テーブルを生成できます。

さまざまなテーブル（つまり、DDM）を 1 つの FROM 節にアドレス指定できるだけでなく、*subqueries* を指定した場合は *table-expression* に複数の FROM 節を含めることができるため、関連する基準データベースの識別には、式全体の最初の FROM 節に指定した最初の DDM のデータベース ID（DBID）が使用されます。

table-name にはオプションで *correlation-clause* を割り当てることができます。*subquery* の場合は、*correlation-clause* を割り当てる必要があります。

ジョインテーブル

joined-table では、「ジョイン」 処理による中間テーブルを指定します。

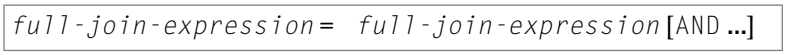
「ジョイン」 には、INNER、LEFT OUTER、RIGHT OUTER、または FULL OUTER JOIN を使用できます。 何も指定しないと、INNER が有効になります。

複数の 「ジョイン」 処理をネストできます。つまり、中間結果テーブルを作成するテーブル自体を JOIN（ジョイン）処理または *subquery* の中間結果テーブルにすることができます。次に後者に *joined-table* を含めたり、その FROM 節に他の *subquery* を指定したりすることもできます。

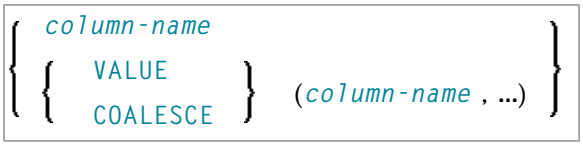
ジョイン条件

INNER、LEFT OUTER、および RIGHT OUTER ジョインの場合：

FULL OUTER ジョインの場合：



フルジョイン式



join-expression 内には、*column-names* および *scalar-function* VALUE（または同義語 COALESCE）のみを使用できます。 詳細については、*column-name* を参照してください。

WHERE 節

```
[WHERE search-condition]
```

WHERE 節を使用して行に対する選択条件（*search-condition*）を指定します。

例：

```
DEFINE DATA LOCAL
01 NAME   (A20)
01 AGE    (I2)
END-DEFINE
...
SELECT *
  INTO NAME, AGE
  FROM SQL-PERSONNEL
  WHERE AGE = 32
END-SELECT
...
```

詳細については、*search-condition* を参照してください。

GROUP BY 節

```
[GROUP BY column-reference,...]
```

GROUP BY 節では、FROM 節に指定されたテーブルをグループに再配列します。これで、各グループ内のすべての行は GROUP BY 列に対して同じ値を持ちます。

選択リストの各 *column-reference* は、GROUP BY 列にするか、または *aggregate-function* 内に指定する必要があります。 集積関数は個々のグループ（テーブル全体ではない）に適用されます。 結果テーブルにはグループと同じ数の行が含まれます。

詳細については、*column-reference* および *aggregate-function* を参照してください。

例：

```
DEFINE DATA LOCAL
1 #AGE     (I2)
1 #NUMBER  (I2)
END-DEFINE
...
SELECT AGE , COUNT(*)
  INTO #AGE, #NUMBER
  FROM SQL-PERSONNEL
```

```
  GROUP BY AGE
  ...
```

GROUP BY 節の前に WHERE 節を指定した場合、グループ化が行われる前に WHERE 節を満たさないすべての行が削除されます。

HAVING 節

```
[HAVING search-condition]
```

HAVING 節を指定する場合、GROUP BY 節も指定する必要があります。

WHERE 節を使用して結果テーブルから行を削除する場合のように、HAVING 節では *search-condition* を基準にしてグループを削除します。 HAVING 節の *Scalar-expressions* はグループごとに単一の値にする必要があります。

詳細については、*scalar-expression* および *search-condition* を参照してください。

例：

```
DEFINE DATA LOCAL
1 #NAME     (A20)
1 #AVGAGE   (I2)
1 #NUMBER   (I2)
END-DEFINE
...
SELECT NAME, AVG(AGE), COUNT(*)
  INTO #NAME, #AVGAGE, #NUMBER
  FROM SQL-PERSONNEL
  GROUP BY NAME
  HAVING COUNT(*) > 1
  ...
```

144 フレキシブル SQL

このchapterでは、次のトピックについて説明します。

フレキシブル SQL の使用

SQL 構文（上記のセクションを参照）に加えて、フレキシブル SQL では任意の SQL 構文を使用できます。

文字 "<<" および ">>"

フレキシブル SQL は "<<" および ">>" 文字で囲まれます。 任意の SQL テキストおよびホスト変数を指定できます。フレキシブル SQL 内では、ホスト変数の先頭にコロン（:）を付ける*必要があります。*

フレキシブル SQL 文字列は複数のステートメント行にわたって指定することもできます。 コメントも指定できます（`PROCESS SQL` ステートメントも参照）。

フレキシブル SQL は、次の SQL 構文項目と置き換えて使用できます。

- *atom*
- *column-reference*
- *scalar-expression*
- *predicate*

フレキシブル SQL は、選択式の節間でも使用できます。

```
SELECT selection
  << ... >>
  INTO ...
  FROM ...
  << ... >>
  WHERE ...
  << ... >>
  GROUP BY ...
  << ... >>
  HAVING ...
  << ... >>
  ORDER BY ...
  << ... >>
```

Note: フレキシブル SQL に使用される SQL テキストは Natural コンパイラには認識されません。 SQL テキスト（置き換えられたホスト変数とともに）は単に SQL 文字列にコピーされ、データベースシステムに渡されます。 フレキシブル SQL の構文エラーは、データベースで対応するステートメントが実行されるときにランタイムに検出されます。

例 **1**：

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE << MONTH (BIRTH) >>  =  << MONTH (CURRENT_DATE) >>
```

例 **2**：

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE << MONTH (BIRTH) = MONTH (CURRENT_DATE) >>
```

例 **3**：

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE SALARY > 50000
<< INTERSECT
   SELECT NAME
   FROM SQL-EMPLOYEES
   WHERE DEPT = 'DEPT10'
>>
```

フレキシブル SQL でのテキスト変数の指定

フレキシブル SQL 内では、「テキスト変数」 を指定することもできます。

```
<<:T:host-variable [LINDICATOR:host-variable]>>
```

構文の各項目については、以下で説明します。

:T:	テキスト変数は接頭文字 ":T:" を付加した *host-variable* です。 これは英数字フォーマットにする必要があります。 SQL ステートメント内のテキスト変数は、ランタイムにその内容で置き換えられます。つまり、テキスト変数に含まれるテキスト文字列が SQL 文字列に挿入されるということです。 置き換えの後、挿入されたテキスト文字列から末尾の空白が削除されます。 テキスト変数の内容が構文的に正しい SQL 文字列であることをユーザー自身が確実にする必要があります。 特に、テキスト変数の内容に *host-variables* が含まれないようにする必要があります。

	テキスト変数を含むステートメントは常にダイナミック SQL モードで実行されます。
LINDICATOR	**LINDICATOR** オプション： テキスト変数の後にキーワード LINDICATOR と長さインジケータ変数（先頭にコロンを付加した *host-variable*）を指定できます。 長さインジケータ変数のフォーマット／長さは I2 にする必要があります。 LINDICATOR 変数を指定しない場合、テキスト変数の内容全体が SQL 文字列に挿入されます。 LINDICATOR 変数を指定した場合、テキスト変数の内容の先頭 *n*（*n* は LINDICATOR 変数の値になる）文字のみが SQL 文字列に挿入されます。 LINDICATOR 変数内の数がテキスト変数の内容の長さより大きい場合、テキスト変数の内容全体が挿入されます。 LINDICATOR変数内の数が負の値または 0 の場合は、何も挿入されません。 全般的な情報については、*host-variable* を参照してください。

テキスト変数の使用例

```
DEFINE DATA LOCAL
01 TEXTVAR (A200)
01 TABLES VIEW OF SYSIBM-SYSTABLES
   02 NAME
   02 CREATOR
END-DEFINE
*
MOVE 'WHERE NAME > ''SYS'' AND CREATOR = ''SYSIBM''' TO TEXTVAR
*
SELECT * INTO VIEW TABLES
  FROM SYSIBM-SYSTABLES
  << :T:TEXTVAR >>
  DISPLAY TABLES
END-SELECT
*
END
```

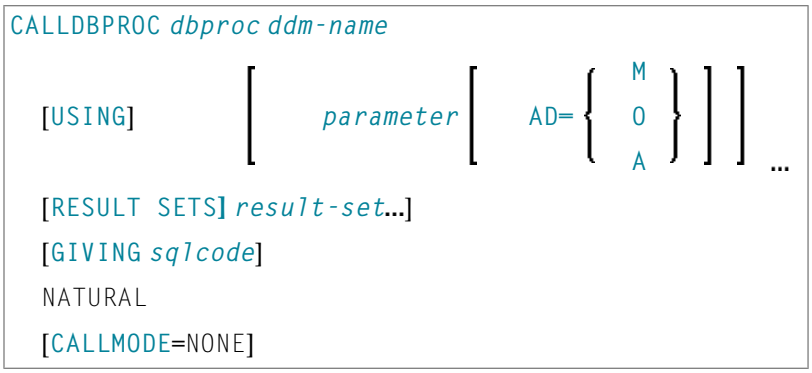
生成される SQL ステートメント（LISTSQL システムコマンドで表示される）は次のようになります。

```
SELECT NAME, CREATOR FROM SYSIBM.SYSTABLES:T: FOR FETCH ONLY
```

実行される SQL ステートメントは次のようになります。

```
SELECT TABNAME, CREATOR FROM SYSIBM.SYSTABLES
  WHERE TABNAME > 'SYS' AND CREATOR = 'SYSIBM'
```

145 CALLDBPROC - SQL



このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

CALLDBPROC ステートメントは、Natural からアクセスできる SQL データベースシステムのストアドプロシージャを呼び出すために使用します。

ストアドプロシージャは、Natural サブプログラムまたは Natural 以外のプログラミング言語で記述されたプログラムのいずれかです。

CALLDBPROC は、呼び出し側オブジェクトとストアドプロシージャ間のパラメータの受け渡しに加えて、「結果セット」 をサポートします。結果セットはストアドプロシージャから呼び出し側オブジェクトへ、パラメータを使用するよりも大量のデータを返すことを可能にします。

結果セットは、ストアドプロシージャによって作成され、呼び出し側オブジェクトで READ RESULT SET ステートメントを使用して読み込んだり処理したりすることのできる 「一時的な結果テーブル」 です。

Note: 一般的に、ストアドプロシージャの呼び出しは Natural サブプログラムの呼び出しと比較されることがあります。CALLDBPROC ステートメントを実行するとき、制御はストアドプロシージャに渡され、ストアドプロシージャの処理後、呼び出し側オブジェクトに戻り、CALLDBPROC ステートメントの次のステートメントに処理が続きます。

構文説明

dbproc	呼び出すストアドプロシージャ： *dbproc* として、呼び出すストアドプロシージャの名前を指定します。その名前は英数字変数または定数（アポストロフィで囲む）として指定できます。 名前はターゲットデータベースシステムのストアドプロシージャの命名規則に従う必要があります。 ストアドプロシージャがNaturalサブプログラムである場合、実際のプロシージャ名は8文字以下である必要があります。	
ddm-name	**Natural** データ定義モジュールの名前： DDMの名前は、ストアドプロシージャを実行するデータベースの「アドレス」を示すように指定する必要があります。詳細については、「*ddm-name*」を参照してください。	
[USING] *parameter*	渡されるパラメータ： *parameter* として、呼び出し側オブジェクトからストアドプロシージャに渡されるパラメータを指定できます。*parameter* には次のものを使用できます。 ■ ホスト変数（オプションで INDICATOR および LINDICATOR 節を使用） ■ 定数 ■ キーワード NULL. 詳細については、*host-variable* を参照してください。	
AD=	属性定義： *parameter* が *host-variable* の場合、次のようにマークできます。	
	AD=O	変更不可。セッションパラメータ AD=O を参照してください。 （DB2 for z/OS での対応するプロシージャ表記：IN）
	AD=M	変更可。セッションパラメータ AD=M を参照してください。 （DB2 for z/OS での対応するプロシージャ表記：INOUT）
	AD=A	入力のみ。セッションパラメータ AD=A を参照してください。 （DB2 for z/OS での対応するプロシージャ表記：OUT）
	parameter が定数の場合は、AD を明示的に指定することはできません。定数には常に AD=O が適用されます。	

result-set	結果セットロケータ変数用のフィールド： *result-set* として、結果セットロケータを返すフィールドを指定します。 結果セットには、フォーマット／長さ I4 の変数を指定する必要があります。 結果セット変数の値は、結果セットを識別し、すぐ次の READ RESULT SET ステートメントで参照できる番号です。 *result-set*の値の順序は、ストアドプロシージャから返される結果セットの順序に対応します。 結果セットの内容は、次の READ RESULT SET ステートメントで処理できます。 結果セットが返されない場合は、対応する結果セット変数に "0" が含まれます。 結果セットは 1 つだけ指定できます。
GIVING *sqlcode*	**GIVING** *sqlcode* オプション： このオプションは、ストアドプロシージャを呼び出す SQL CALL ステートメントの SQL コードを取得するために使用できます。 このオプションを指定し、ストアドプロシージャの SQL コードが "0" でない場合、Natural エラーメッセージは発行されません。 この場合、SQL コード値に対して取られる動作は、呼び出し側の Natural オブジェクト内でコード化されている必要があります。 *sqlcode* フィールドには、フォーマット／長さ I4 の変数を指定する必要があります。 GIVING *sqlcode* オプションを指定せず、ストアドプロシージャの SQL コードが "0" でない場合は、Natural エラーメッセージが発行されます。
CALLMODE=NONE	**CALLMODE** パラメータ Natural for Windows、Natural for UNIX、および Natural for OpenVMS では、CALLMODE を指定しないでください。

例

次の例は、ストアドプロシージャ DEMO_PROC を呼び出して、PERSON テーブルの特定範囲内の名前をすべて取得する Natural プログラムを示しています。

3 つのパラメータフィールドが DEMO_PROC に渡されます。1 番目と 2 番目のパラメータは、名前の範囲の開始値と終了値をストアドプロシージャに渡します。3 番目のパラメータは、条件に一致する名前を受け取ります。

この例では、名前は、READ RESULT SETステートメントで処理される結果セットに返されます。

```
DEFINE DATA LOCAL
1 PERSON VIEW OF DEMO-PERSON
  2 PERSON_ID
  2 LAST_NAME
1 #BEGIN    (A2) INIT <'AB'>
1 #END      (A2) INIT <'DE'>
1 #RESPONSE (I4)
1 #RESULT   (I4)
1 #NAME (A20)
END-DEFINE

...

CALLDBPROC 'DEMO_PROC' DEMO-PERSON #BEGIN (AD=O) #END (AD=O) #NAME (AD=A)
    RESULT SETS #RESULT
    GIVING #RESPONSE

READ RESULT SET #RESULT INTO #NAME FROM DEMO-PERSON
    GIVING #RESPONSE
  DISPLAY #NAME
END-RESULT

...

END
```

146 COMMIT - SQL

```
COMMIT
```

このchapterでは、次のトピックについて説明します。

機能

SQL COMMIT ステートメントは END TRANSACTION ステートメントに対応します。 論理トランザクションの終了を示し、トランザクション中にロックされた全データを解放します。 すべてのデータ変更がコミットされて確定されます。

Important: 作業の論理ユニット終了時には全カーソルがクローズされるため、COMMIT ステートメントはデータベース更新処理ループ内に指定しないでください。更新処理ループの外側、またはネストされたループの一番外側のループの後に指定する必要があります。

例

```
...
DELETE FROM SQL-PERSONNEL WHERE NAME = 'SMITH'
COMMIT
...
```

147 DELETE - SQL

このchapterでは、次のトピックについて説明します。

機能

SQL DELETE ステートメントは、カーソルを使用しないでテーブル内の行を削除したり（「検索済」 DELETE）、カーソルが位置づけられたテーブル内の行を削除したり（「位置決め」 DELETE）するために使用します。

構文説明

2 つの異なる構造が可能です。

- 構文 1 - 検索済 DELETE
- 構文 2 - 位置決め DELETE

構文 1 - 検索済 DELETE

「検索済」 DELETE ステートメントは、どの SELECT ステートメントにも関連しない独立したステートメントです。単一のステートメントで、ゼロ、1つ、複数、またはすべての行をテーブルから削除できます。削除される行はテーブルに適用する *search-condition* によって決定されます。 オプションで、テーブル名に *correlation-name* を割り当てることができます。

Note: 「検索済」 DELETE で実際に削除された行の数は、システム変数 *ROWCOUNT（『システム変数』ドキュメントを参照）で確認できます。

一般セットの構文：

```
DELETE FROM table-name [(correlation-name)] [WHERE search-condition]
```

拡張セットの構文：

```
DELETE FROM table-name [(correlation-name)] [WHERE search-condition]
    [ WITH { RR | RS | CS } ]
    [QUERYNO integer]
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

構文要素の説明：

FROM *table-name*	**FROM** 節： この節では、行を削除するテーブルを指定します。
correlation-name	オプションで、テーブル名に *correlation-name* を割り当てることができます。
WHERE *search-condition*	**WHERE** 節： この節では、削除する行の選択条件を指定します。 WHERE 節を指定しないと、テーブル全体が削除されます。
WITH	**WITH** 分離レベル節： この節は **SQL** 拡張セットに含まれます。 この節では、削除する行を検索するときに使用する分離レベルを明示的に指定できます。 DB2 データベースにのみ有効です。 他のデータベースに使用すると、ランタイムエラーが発生します。
QUERYNO *integer*	**QUERYNO** 節： この節は **SQL** 拡張セットに含まれます。 この節は、現在はサポートされていないため無視されます。

構文 2 - 位置決め DELETE

「位置決め」 DELETE ステートメントは、常にデータベースループ内のカーソルを参照します。したがって、位置決め DELETE ステートメントで参照されるテーブルは、対応する SELECT ステートメントで参照されるテーブルと同一のものである必要があります。そうでない場合はエラーメッセージが返されます。 位置決め DELETE を非カーソル選択で使用することはできません。

機能的に、位置決め DELETE ステートメントは 「通常の」 **Natural** DELETE ステートメントに対応します。

```
DELETE FROM table-name WHERE CURRENT OF CURSOR [(r)]
```

構文図で使用されている記号については、「*構文記号*」を参照してください。

構文要素の説明：

FROM *table-name* **WHERE CURRENT OF CURSOR**	**FROM** 節： この節では、行を削除するテーブルを指定します。
(*r*)	ステートメント参照： (*r*) 表記を使用して、削除する行の選択に使用されたステートメントを参照します。 ステートメント参照の指定がない場合、DELETE ステートメントは、データベースレコードが選択された一番内側のアクティブな処理ループを参照します。

148 INSERT - SQL

一般セットの構文：

```
INSERT INTO table-name { (*) [VALUES-clause]
                       [(column-list)] VALUE-LIST }
```

拡張セットの構文：

```
INSERT INTO table-name { (*) [OVERRIDING USER VALUE] [VALUES-clause]
                       [(column-list)] [OVERRIDING USER VALUE] VALUE-LIST }
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

SQL INSERT ステートメントは、1 つ以上の行をテーブルに新しく追加するために使用します。

構文説明

INTO *table-name*	**INTO** 節： INTO 節では、新しい行を挿入するテーブルを指定します。 詳細については、 *table-name* を参照してください。
column-list	列リスト： 構文： `column-name...` *column-list* には、1 つ以上の *column-names* を指定できます。これは現在挿入される行内の値とともに指定します。 *column-list* を指定する場合、列の順序は *insert-item-list* に指定される値の順序または指定ビュー（下記参照）に含まれる値の順序と一致している必要があります。 *column-list* を指定しない場合、*insert-item-list* または指定ビュー内の値がすべての列の暗黙のリストに従ってテーブルに存在する順序で挿入されます。
VALUES-clause	**VALUES** 節：

	VALUES 節を使用して、*1* つの行をテーブルに挿入します。 下記の「*VALUES 節*」を参照してください。
insert-item-list	**INSERT 1** 行： *insert-item-list* には、*column-list* に指定された列に割り当てる 1 つ以上の値を指定できます。 指定する値の順序が列の順序と一致している必要があります。 *column-list* を指定しない場合は、*insert-item-list* 内の値がすべての列の暗黙のリストに従ってテーブルに存在する順序で挿入されます。 *insert-item-list* に指定できる値は、*constants*、*parameters*、*special-registers*、または NULL です。 *view-name*、*constant*、および *parameter* については、「*基本構文項目*」を参照してください。 *special-register* の情報も参照してください。 値 NULL が割り当てられている場合は、指定されたフィールドが値（値 "0" または「空白」でも）を受け取らないことを意味します。 例 - INSERT 1 行： `...` `INSERT INTO SQL-PERSONNEL (NAME,AGE)` `  VALUES ('ADKINSON',35)` `...`
OVERRIDING USER VALUE	**OVERRIDING USER VALUE** 節： この節は **SQL 拡張セット**に含まれます。 この節は現在はサポートされていません。 使用すると、コンパイラエラーが発生します。

VALUES 節

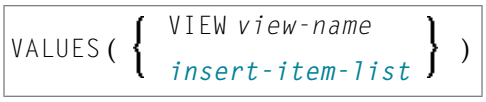
VALUES 節を使用して、*1* つの行をテーブルに挿入します。 アスタリスク（*）が指定されているか、または *column-list* が指定されているかによって、VALUES 節の形式は次のいずれかになります。

アスタリスク指定の **VALUES** 節

```
VALUES (VIEW view-name)
```

アスタリスク表記を指定する場合、VALUES 節にビューを指定する*必要があります*。 このビューのフィールド値により、ビューのフィールド名を列名として使用して、新しい行が指定したテーブルに挿入されます。

列リスト指定の **VALUES** 節

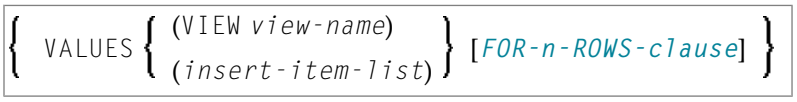


column-list を指定し、VALUES 節でビューを参照する場合、列リストに指定する項目数は、*VALUE-LIST* 内のビューに定義されたフィールド数に対応する必要があります。

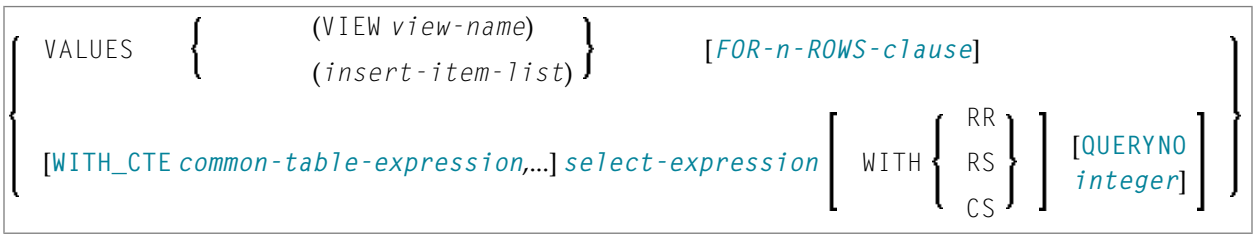
column-list を指定しない場合は、ビューに定義されたフィールドが、すべての列の暗黙のリストに従って、指定したテーブルに存在する順序で挿入されます。

VALUE-LIST

一般セットの構文：



拡張セットの構文：



構文の説明：

VIEW *view-name*	このビューのフィールド値により、ビューのフィールド名を列名として使用して、新しい行が指定したテーブルに挿入されます。
insert-item-list	**INSERT 1** 行： *insert-item-list* には、列リストに指定された列に割り当てる 1 つ以上の値を指定できます。指定する値の順序が列の順序と一致している必要があります。 *column-list* を指定しない場合は、*insert-item-list* 内の値がすべての列の暗黙のリストに従ってテーブルに存在する順序で挿入されます。 *insert-item-list* に指定できる値は、定数、パラメータ、特別レジスタ、または NULL です。 *view-name*、*constant*、および *parameter* については、「*基本構文項目*」を参照してください。特別レジスタの情報も参照してください。 値 NULL が割り当てられている場合は、指定されたフィールドが値（値 0 または 空白でも）を受け取らないことを意味します。

	例 - INSERT 1 行： `...` `INSERT INTO SQL-PERSONNEL (NAME,AGE)` `VALUES ('ADKINSON',35)` `...`
FOR-n-ROWS-clause	オプションの節（下記参照）
WITH_CTE *common-table-expression*	この節は **SQL** 拡張セットに含まれます。 このオプションの節では、後に続く SELECT ステートメントの任意の FROM 節で参照できる結果テーブルを定義できます。単一の WITH_CTE キーワードに続けて複数の共通テーブル式を指定できます。それぞれの共通テーブル式は、後続の共通テーブル式の FROM 節でも参照できます。 詳細については、「*SELECT - カーソル指向（[WITH CTE common-table-expression,...] 構文）*」を参照してください。
select-expression	複数行の **INSERT**： この節は **SQL** 拡張セットに含まれます。 *select-expression* では、*multiple*行をテーブルに挿入します。*select-expression*が評価されると、結果テーブルの各行は、単一行の INSERT 処理の ***VALUES** 節*の値として行に値が指定された場合と同様に扱われます。 詳細については、「*選択式*」を参照してください。 例 - 複数行の INSERT： `...` `INSERT INTO SQL-RETIREE (NAME,AGE,SEX)` `  SELECT LASTNAME, AGE, SEX` `  FROM SQL-EMPLOYEES` `  WHERE AGE > 60` `...` **注意:** 実際に挿入された行の数は、システム変数 *ROWCOUNT（『システム変数』ドキュメントを参照）で確認できます。
WITH RR/RS/CS	**WITH** 分離レベル節： この節は **SQL** 拡張セットに含まれます。 この節では、挿入する行を検索するときに使用する分離レベルを明示的に指定できます。 DB2 データベースにのみ有効です。 他のデータベースに使用すると、ランタイムエラーが発生します。

QUERYNO_*integer*	**QUERYNO** 節： この節は **SQL 拡張セット**に含まれます。 この節は、現在はサポートされていないため無視されます。

FOR-*n*-ROWS-Clause

```
FOR { [:]_host-variable } ROWS [ { ATOMIC                                  } ]
    { integer          }       [ { NOT ATOMIC CONTINUE ON SQLEXCEPTION } ]
```

この節は次の副節で構成されます。

FOR [:] *hostvariable*/*integer* **ROWS** 節

```
FOR { [:]_host-variable } ROWS
    { integer          }
```

この節の指定はオプションです。 指定する必要があるのは、次の場合のみです。

- コンパイラオプション DB2ARRY を指定している。
- 複数の行が ***VALUES** 節*の *insert-item-list* で指定した配列から挿入される。

指定する場合、[:]_*hostvariable*/*integer* では、*VALUES 節*の *insert-item-list* に指定した配列から DB2 テーブルに挿入する行の数を決定します（最初のオカレンスから挿入します）。

この節により、ループ内でNatural配列から行を挿入するプログラムのパフォーマンスが向上します。 この節を使用すると、配列内の行を 1 つの SQL ステートメントで挿入できます。

下記の例を参照してください。

ATOMIC 節

```
{ ATOMIC                                  }
{ NOT ATOMIC CONTINUE ON SQLEXCEPTION }
```

この節では、複数の行の挿入を DB2 で原子処理として扱うかどうかを指定します。

指定する必要があるのは、次の場合のみです。

- コンパイラオプション DB2ARRY を指定している。
- 複数の行が ***VALUES** 節*の *insert-item-list* で指定した配列から挿入される。

構文の説明：

ATOMIC	エラーが発生した場合は、ターゲットテーブルに行を挿入しないことを指定します。 これはデフォルト値です。
NOT ATOMIC CONTINUE ON SQLEXCEPTION	エラーが発生した場合は、エラーのないすべての行を挿入し、エラーのある行を DB2 で破棄することを指定します。

このような場合に返される SQL コードについては、DB2 SQL REFERENCE を参照してください。

例

```
DEFINE DATA LOCAL
01 NAME        (A20/1:10)  INIT <'ZILLER1','ZILLER2','ZILLER3','ZILLER4'
                           ,'ZILLER5','ZILLER6','ZILLER7','ZILLER8'
                           ,'ZILLER9','ZILLERA'>
01 ADDRESS     (A100/1:10) INIT <'ANGEL STREET 1','ANGEL STREET 2'
                           ,'ANGEL STREET 3','ANGEL STREET 4'
                           ,'ANGEL STREET 5','ANGEL STREET 6'
                           ,'ANGEL STREET 7','ANGEL STREET 8'
                           ,'ANGEL STREET 9','ANGEL STREET 10'>
01 DATENATD (D/1:10)  INIT <D'1954-03-27',D'1954-03-27',D'1954-03-27'
                          ,D'1954-03-27',D'1954-03-27',D'1954-03-27'
                          ,D'1954-03-27',D'1954-03-27',D'1954-03-27'
                          ,D'1954-03-27'>
01 SALARY      (P4.2/1:10) INIT <1000,2000,3000,4000,5000
                           ,6000,7000,8000,9000,9999>
01 L§ADDRESS   (I2/1:10) INIT <14,14,14,14,14,14,14,14,14,15>
01 N§ADDRESS   (I2/1:10) INIT <00,00,00,00,00,00,00,00,00,00>
01 ROWS        (I4)
01 INDEX       (I4)
01 V1 VIEW OF NAT-DEMO_ID
02 NAME
02 ADDRESS     (EM=X(20))
02 DATEOFBIRTH
02 SALARY
01 ROWCOUNT  (I4)
END-DEFINE
OPTIONS DB2ARRY=ON                  /* <-- ENABLE DB2 ARRAY
ROWCOUNT := 10
INSERT INTO NAT-DEMO_ID
       (NAME,ADDRESS,DATEOFBIRTH,SALARY)
       VALUES
       (:NAME(*),                   /* <-- ARRAY
        :ADDRESS(*)                 /* <-- ARRAY
        INDICATOR :N§ADDRESS(*)     /* <-- ARRAY
```

```
        LINDICATOR :L§ADDRESS(*),   /* <-- ARRAY DB2 VCHAR
        :DATENATD(1:10),            /* <-- ARRAY NATURAL DATES
        :SALARY(01:10)              /* <-- ARRAY NATURAL PACKED
       )
       FOR :ROWCOUNT ROWS
SELECT * INTO VIEW V1 FROM NAT-DEMO_ID WHERE NAME > 'Z'
DISPLAY V1                          /* <-- VERIFY INSERT
END-SELECT
END
```

149 PROCESS SQL

```
PROCESS SQL ddm-name <<statement-string>>
```

このchapterでは、次のトピックについて説明します。

機能

PROCESS SQL ステートメントは、基準データベースに対して SQL ステートメントを発行するために使用します。

構文説明

ddm-name	DDM の名前は、ストアドプロシージャを実行するデータベースの「アドレス」を示すように指定する必要があります。 詳細については、「*ddm-name*」を参照してください。
statement-string	*statement-string*に指定できるステートメントは、SQL ステートメント EXECUTE で発行できるステートメントと同じです（「*フレキシブル SQL*」も参照）。 **注意:** Natural 環境と基準データベースのトランザクション同期の問題を避けるために COMMIT ステートメントおよび ROLLBACK ステートメントを PROCESS SQL 内で使用しないでください。 ステートメント文字列は継続文字を指定しなくても複数のステートメント行にわたって指定できます。行の末尾にコメントを追加することも、行全体をコメント行にすることもできます。 ステートメント文字列にはパラメータを含めることもできます。下記の「パラメータ」を参照してください。

パラメータ

```
[ :U ]
[ :G ]  :host-variable [INDICATOR:host-variable] [LINIDICATOR:host-variable]
```

説明した *parameter* と異なり、ここで述べる *host-variables* には先頭にコロン（:）を付ける必要があります。 さらに、修飾子（":U" または ":G"）を付けることができます。

詳細については、*host-variable* を参照してください。

構文要素の説明：

:U:*host-variable*	接頭文字 ":U" は 「Using」 変数と呼ばれるホスト変数を修飾します。 この変数は、値がデータベースに渡されることを示します。 ":U" はデフォルト指定です。
:G:*host-variable*	接頭文字 ":G" は 「Giving」 変数と呼ばれるホスト変数を修飾します。 この変数は、データベースから値を受け取ることを示します。

Entire Access オプション

Entire Access では、*statement-string* として次を指定することもできます。

- SET SQLOPTION *option = value*
- SQLCONNECT *option = value*
- SQLDISCONNECT

これらのオプションは Entire Access でのみ使用できます。「*SQL データベースのデータへのアクセス*」（『*プログラミングガイド*』）を参照してください。

例

Adabas D の例：

```
PROCESS SQL ADABAS_D_DDM << LOCK TABLE EMPLOYEES IN SHARE MODE >>
```

Adabas D に保存されたプロシージャの呼び出し例：

呼び出されたプロシージャは、2 つの数字の合計を計算します。

```
...
COMPUTE #N1 = 1
COMPUTE #N2 = 2
COMPUTE #SUM = 0
...
PROCESS SQL ADABAS_D_DDM << DBPROCEDURE DEMO.SUM (:#N1, :#N2, :G:#SUM) >>
...
WRITE #N1 '+' #N2 ' =' #SUM
...
```

150 READ RESULT SET - SQL

一般セットの構文：

```
READ [(limit)] RESULT SET result-set INTO { VIEW view-name } FROM ddm-name
                                          { parameter     }
  [GIVING [:] sql-code]
END-RESULT
```

拡張セットの構文：

```
READ [(limit)] RESULT SET result-set { VIEW view-name } FROM ddm-name
INTO                                 { parameter     }
[WITH INSENSITIVE SCROLL [:] scroll-hv]
  [GIVING [:] sql-code]

                                       integer
END-RESULT
```

このchapterでは、次のトピックについて説明します。

構文図で使用されている記号については、「*構文記号*」を参照してください。

機能

SQL ステートメント READ RESULT SET は、CALLDBPROC ステートメントと組み合わせてのみ使用できます。 前の CALLDBPROC ステートメントで呼び出されたストアドプロシージャによって作成された結果セットを読み込むために使用します。

構文説明

limit	読み込む行の数を制限できます。 数値定数（0～4294967295）または N、P または I フォーマットの変数として制限を指定できます。
result-set	結果セットとして、先行する CALLDBPROC ステートメントで代入された結果セットロケータ変数を指定します。 結果セットには、フォーマット／長さ I4 の変数を指定する必要があります。 **注意:** CALLDBPROC ステートメントと READ RESULT SET ステートメント間の同期地点の処理が行われる場合、結果セットには READ RESULT SET ステートメントでアクセスできなくなります。

FROM *ddm-name*	*ddm-name*として、ストアドプロシージャを実行するデータベースの「アドレス指定」に使用するDDMの名前を指定します。詳細については、「*ddm-name*」を参照してください。
WITH INSENSITIVE SCROLL [:] *scroll_hv*	この節は **SQL 拡張セット**に含まれます。 この節は現在はサポートされていません。 使用すると、コンパイラエラーが発生します。
GIVING *sqlcode*	このオプションは、結果セットの処理に使用した SQL 「フェッチ」 操作の SQL コードを取得するために使用できます。 このオプションを指定し、SQL 操作の SQL コードが "0" でない場合、Natural エラーメッセージは発行されません。 この場合、SQL コード値に対して取られる動作は、呼び出し側の Natural オブジェクト内でコード化されている必要があります。 *sqlcode*フィールドには、フォーマット／長さ I4 の変数を指定する必要があります。 GIVING *sqlcode*オプションを指定せず、SQL コードが "0" でない場合は、Natural エラーメッセージが発行されます
END-RESULT	READ RESULT SET ステートメントを終了するには、Natural 予約キーワード END-RESULT を使用する必要があります。

例

CALLDBPROC ステートメントの例を参照してください。

151 ROLLBACK - SQL

ROLLBACK

このchapterでは、次のトピックについて説明します。

機能

SQL ROLLBACK ステートメントは Natural BACKOUT TRANSACTION ステートメントに対応します。最後のリカバリ単位の開始以降に行われたデータベース更新をすべて元に戻します。 リカバリ単位は、セッションの開始後、または最後の SYNCPOINT、COMMIT、END TRANSACTION、または BACKOUT TRANSACTION ステートメントの後に開始できます。 このステートメントも、トランザクション中に保持されていたすべてのレコードを解放します。

端末I/Oによりすでにコミットされている更新をプログラムでバックアウトしようとした場合、対応する Natural エラーメッセージ（NAT3711）が返されます。

Caution: 作業の論理ユニット終了時には全カーソルがクローズされるため、ROLLBACK ステートメントはデータベース更新処理ループ内に指定しないでください。更新処理ループの外側、またはネストされたループの一番外側のループの後に指定する必要があります。

Natural 以外のプログラムについて

Naturalプログラムから他の標準プログラミング言語で記述された外部プログラムを呼び出す場合、Naturalプログラムでもデータベースコールを発行するのであれば、外部プログラムに独自の ROLLBACK ステートメントが含まれないようにする必要があります。 呼び出し側の Natural プログラムが外部プログラムに代わって ROLLBACK ステートメントを発行する必要があります。

例

```
...
DELETE FROM SQL-PERSONNEL WHERE NAME = 'SMITH'
ROLLBACK
...
```

152 SELECT - SQL

このchapterでは、次のトピックについて説明します。

機能

SELECT ステートメントでは、任意の行数を取得するためのカーソル選択と、多くても 1 つの行を取得する非カーソル選択（単独 SELECT）の両方がサポートされています。

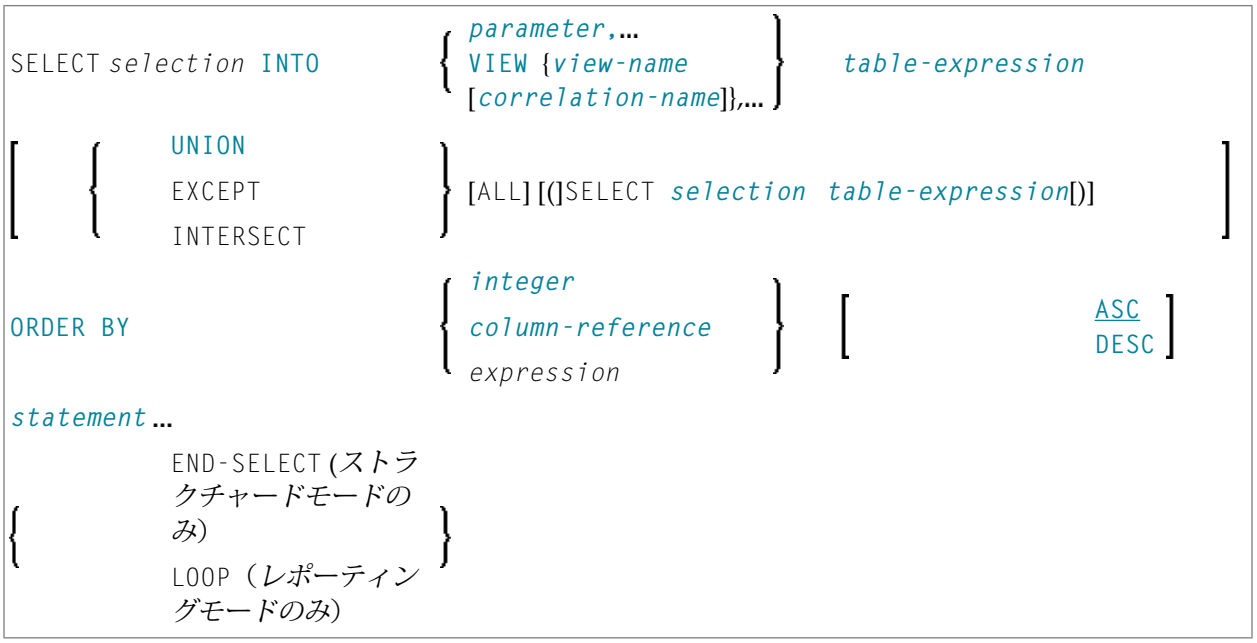
SELECT ... END-SELECT 構造で、Natural は FIND ステートメントと同じデータベースループ処理を使用します。

構文説明

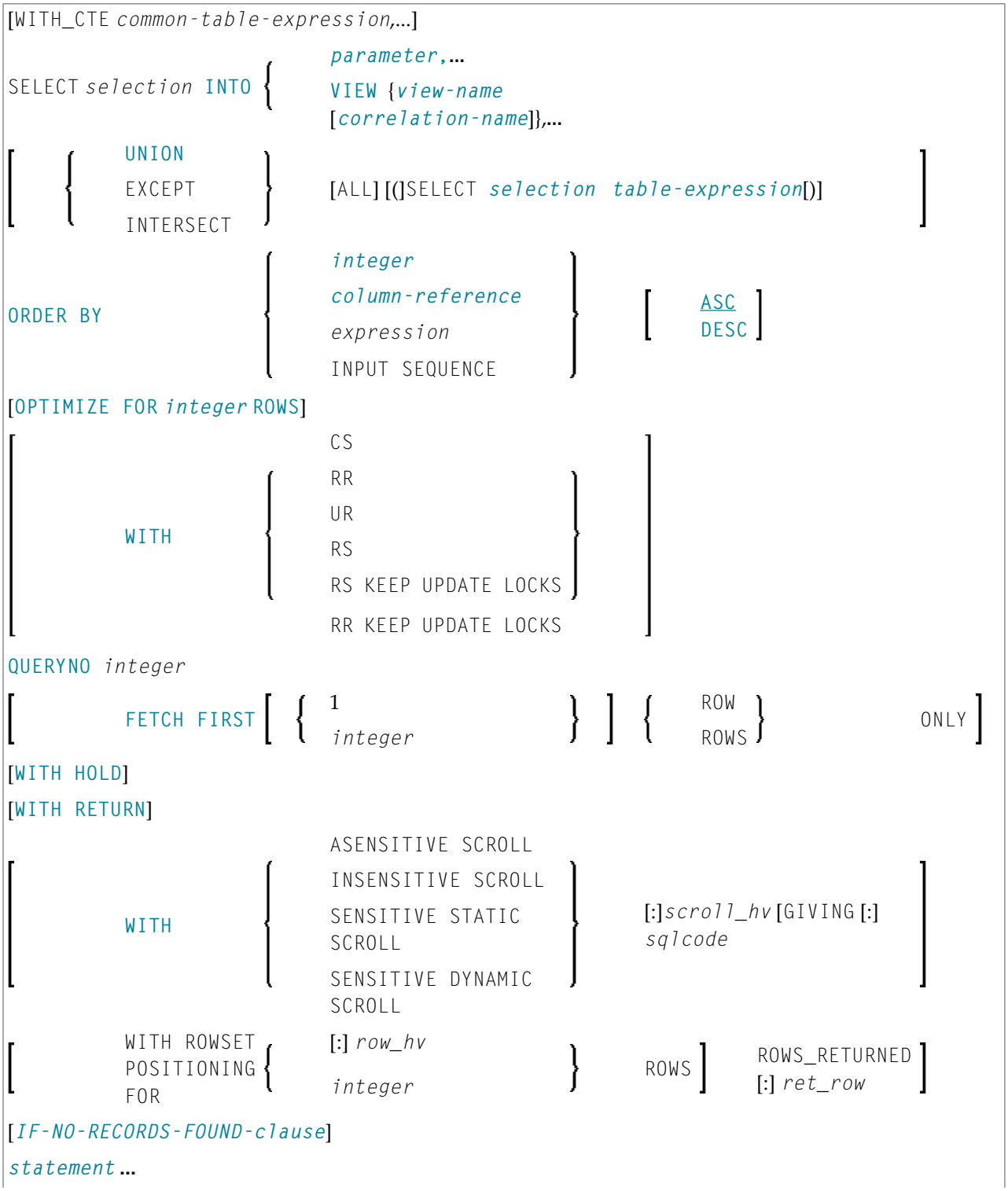
2 つの異なる構造が可能です。

構文 **1 -** カーソル選択

一般セットの構文



拡張セットの構文：


[WITH_CTE common-table-expression,...]
SELECT selection INTO
parameter,...
VIEW {view-name
[correlation-name]},...
UNION
EXCEPT
INTERSECT
[ALL] [(]SELECT selection table-expression[)]
ORDER BY
integer
column-reference
expression
INPUT SEQUENCE
ASC
DESC
[OPTIMIZE FOR integer ROWS]
WITH
CS
RR
UR
RS
RS KEEP UPDATE LOCKS
RR KEEP UPDATE LOCKS
QUERYNO integer
FETCH FIRST
1
integer
ROW
ROWS
ONLY
[WITH HOLD]
[WITH RETURN]
WITH
ASENSITIVE SCROLL
INSENSITIVE SCROLL
SENSITIVE STATIC SCROLL
SENSITIVE DYNAMIC SCROLL
[:]scroll_hv [GIVING [:] sqlcode
WITH ROWSET POSITIONING FOR
[:] row_hv
integer
ROWS
ROWS_RETURNED [:] ret_row
[IF-NO-RECORDS-FOUND-clause]
statement ...

```
                END-SELECT
                (ストラク
                チャードモー
{               ドのみ)      }
                LOOP（レポー
                ティングモー
                ドのみ)
```

構文要素の説明 - 構文 1：

SELECT *selection*	FIND ステートメントと同様、カーソル選択は検索条件を基準に 1 つ以上のデータベーステーブルから行（レコード）のセットを選択するために使用します。また、アプリケーションプログラムからのカーソル管理は必要なく、Natural によって自動的に処理されます。 詳細については、以下の「***SELECT - カーソル指向***」を参照してください。
INTO	INTO 節は、選択の結果を格納するプログラム内のターゲットフィールドを指定するために使用します。詳細と例については、以下の「***INTO 節***」を参照してください。
VIEW	INTO 節で 1 つまたは複数のビューを参照する場合は、*selection* で指定した項目数が、ビューに定義されているフィールド（カウントするグループフィールドではなく、再定義フィールドとインジケータフィールド）の数と一致する必要があります。 詳細と例については、以下の「***VIEW 節***」を参照してください。
table-expression	*table-expression* は、FROM 節とオプションの WHERE 節から成ります。 詳細と例については、以下の「*table-expression*」を参照してください。
UNION	UNION は、複数の *select-expressions* の結果を結合します。 詳細と例については、以下の「***UNION を伴うクエリ***」を参照してください。
ORDER BY	ORDER BY 節は、SELECT ステートメントの結果を特定の順序で整列します。 詳細と例については、以下の「***ORDER BY 節***」を参照してください。
IF NO RECORDS FOUND	IF NO RECORDS FOUND 節は、先行する SELECT ステートメントで指定した選択条件に一致するレコードがない場合に処理ループを開始するために使用します。詳細については、以下の「***IF NO RECORDS FOUND 節***」を参照してください。
END-SELECT	SELECT ステートメントを終了するには、Natural の予約キーワード END-SELECT を使用する必要があります。

次の構文要素は **SQL 拡張セット**に含まれます。

OPTIMIZE FOR	**OPTIMIZE FOR** 節： この節は DB2 データベースにのみ有効です。 他のデータベースに使用すると、ランタイムエラーが発生します。
WITH CS/RS/UR/...	**WITH CS/RS/UR/...**節： この節は DB2 データベースにのみ有効です。 他のデータベースに使用すると、ランタイムエラーが発生します。

QUERYNO	**QUERYNO** 節： この節は、現在はサポートされていないため無視されます。
FETCH FIRST	**FETCH FIRST** 節： この節はDB2データベースにのみ有効です。 他のデータベースに使用すると、ランタイムエラーが発生します。
WITH HOLD	**WITH HOLD** 節： この節は現在サポートされていません。 使用すると、コンパイラエラーが発生します。
WITH RETURN	**WITH RETURN** 節： この節は現在はサポートされていません。 使用すると、コンパイラエラーが発生します。
WITH ... SCROLL	**WITH ... SCROLL** 節： この節はSQL拡張セットに含まれます。 この節では、RDBMSのスクロール可能カーソルを有効にします。 スクロール可能カーソルには、`ASENSITIVE`、`INSENSITIVE`、`SENSITIVE STATIC`、または `SENSITIVE DYNAMIC` を使用できます。 **注意:** すべてのSQLデータベースシステムですべてのオプションがサポートされているわけではありません。 ■ `WITH ASENSITIVE SCROLL` では、カーソルを `INSENSITIVE` または `SENSITIVE DYNAMIC` に指定します。 どちらが使用されるかは、カーソルのオープン時にその読み取り専用プロパティに基づいてデータベースによって決定されます。カーソルが読み取り専用の場合、カーソルは `INSENSITIVE` になります。 カーソルが読み取り専用でない場合、カーソルは `SENSITIVE DYNAMIC` になります。 これは、DB2 データベース用にサポートされています。 ■ `WITH INSENSITIVE SCROLL` では、カーソルの更新後、基本テーブルを対象に実行される更新、削除、および挿入に対してカーソルを非センシティブにすることを指定します。 `INSENSITIVE SCROLL` カーソルに対しては、位置決め UPDATE および位置決め DELETE は許可されていません。 これは、Oracle、Adabas D、Informix、MS SQL Server ODBC、および DB2 データベース用にサポートされています。 ■ `WITH SENSITIVE STATIC` では、カーソルのオープン後、基本テーブルを対象に実行される更新および削除（挿入は含まない）に対してカーソルをセンシティブにすることを指定します。 `SENSITIVE STATIC SCROLL` カーソルに対しては、位置決め UPDATE および位置決め DELETE が許可されています。 これは、Adabas D、MS SQL Server ODBC、および DB2 データベース用にサポートされています。 ■ `WITH SENSITIVE DYNAMIC` では、カーソルのオープン後、基本テーブルを対象に実行される更新、削除、および挿入に対してカーソルをセンシティブにすることを指定します。 `SENSITIVE DYNAMIC SCROLL` カーソルに対しては、位置決め UPDATE および位置決め DELETE が許可されています。 これは、Adabas D、MS SQL Server ODBC、および DB2 データベース用にサポートされています。 スクロール可能カーソルにより、カーソルがオープンされている限り、アプリケーションでカーソル位置の任意の行をいつでも位置決めできるようになります。

	スクロール可能カーソルは、Sybase データベースではサポートされていません。スクロール可能カーソルは、MS SQL Server DBLIB インターフェイスではサポートされていませんが、MS SQL Server ODBC インターフェイスに限ってサポートされています。 位置決めは *scroll_hv* の内容に基づいて行われます。内容は、データベースに対する FETCH が実行されるたびに評価されます。 詳細については、「***SELECT - カーソル指向***」を参照してください。

構文 **2 -** 非カーソル選択

一般セットの構文

```
SELECT SINGLE
  selection INTO { parameter ,...                          } table-expression
                 { VIEW {view-name [correlation-name ]}, ... }
  [IF-NO-RECORDS-FOUND-clause]
  statement...
{ END-SELECT (structured mode only) }
{ LOOP (reporting mode only)        }
```

拡張セットの構文

```
SELECT SINGLE
  selection INTO { parameter ,...                          } table-expression
                 { VIEW {view-name [correlation-name ]}, ... }
  [ WITH { CS } ]
         { RR }
         { UR }
  [IF-NO-RECORDS-FOUND-clause]
  statement...
{ END-SELECT (structured mode only) }
{ LOOP (reporting mode only)        }
```

構文要素の説明 - 構文 2：

SELECT SINGLE	SELECT SINGLE ステートメントでは、非カーソル選択（単独 SELECT）の機能、つまり、カーソルを使用せずに多くても 1 つの行を取得する選択式がサポートされています。 これは位置決め UPDATE や位置決め DELETE ステートメントでは参照できません。
INTO	INTO 節は、選択の結果を格納するプログラム内のターゲットフィールドを指定するために使用します。 詳細と例については、以下の「*INTO 節*」を参照してください。
VIEW	INTO 節で 1 つまたは複数のビューを参照する場合は、*selection* で指定した項目数が、ビューに定義されているフィールド（カウントするグループフィールドではなく、再定義フィールドとインジケータフィールド）の数と一致する必要があります。 詳細と例については、以下の「*VIEW 節*」を参照してください。
table-expression	*table-expression* は、FROM 節とオプションの WHERE 節から成ります。 詳細と例については、以下の「*table-expression*」を参照してください。
WITH CS/RR/UR	**WITH CS/RR/UR** 節： この節は DB2 データベースにのみ有効です。 他のデータベースに使用すると、ランタイムエラーが発生します。
IF NO RECORDS FOUND	IF NO RECORDS FOUND 節は、先行する SELECT ステートメントで指定した選択条件に一致するレコードがない場合に処理ループを開始するために使用します。 詳細については、以下の「*IF NO RECORDS FOUND 節*」を参照してください。
END-SELECT	SELECT ステートメントを終了するには、Natural の予約キーワード END-SELECT を使用する必要があります。

INTO 節

```
INTO { parameter ,...                                  }
     { VIEW {view-name [correlation-name ]},...        }
```

INTO 節は、選択の結果を格納するプログラム内のターゲットフィールドを指定するために使用します。 INTO 節では、DEFINE DATA ステートメントで定義された個々の *parameters* または 1 つ以上のビューを指定できます。

全ターゲットフィールドは単一テーブルまたはジョイン処理でできた複数のテーブルから値を受け取ることができます（「*ジョインクエリ*」も参照）。

Note: 標準 SQL 構文では、INTO 節は非カーソル選択処理（単独 SELECT）でのみ使用されるため、単一行を選択する場合にのみ指定できます。 ただし、Natural では、カーソル選択処理および非カーソル選択処理のどちらにも INTO 節を使用します。

selection はアスタリスク（*）のみでも構成できます。 標準選択式では、FROM 節に指定されたテーブル（複数可）の全列名のリストに対する省略形です。 ただし、同じ構文項目 SELECT * は、Natural SELECT ステートメントでは異なる意味になります。 INTO 節にリストされた項目は

すべて選択にも使用されます。 それらの名前は既存のデータベースの列名に対応させる必要があります。

例：

例 1：

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
...
SELECT *
  INTO NAME, AGE
```

例 2：

```
...
SELECT *
  INTO VIEW PERS
```

上記の例は下記の例と同等です。

例 3：

```
...
SELECT NAME, AGE
  INTO NAME, AGE
```

例 4：

```
...
```

```
SELECT NAME, AGE
  INTO VIEW PERS
```

VIEW 節

```
VIEW {view-name [correlation-name]}, ...
```

INTO 節で 1 つまたは複数のビューを参照する場合は、*selection* で指定した項目数が、ビューに定義されているフィールド（カウントするグループフィールドではなく、再定義フィールドとインジケータフィールド）の数と一致する必要があります。

Note: Natural ターゲットフィールドおよびテーブルの列は両方とも Natural DDM に定義されている必要があります。 ただし、割り当ては順序に従って行われるため、名前が異なっていてもかまいません。

ビュー使用の INTO 節の例：

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
...
SELECT FIRSTNAME, AGE
  INTO VIEW PERS
  FROM SQL-PERSONNEL
...
```

ターゲットフィールド NAME および AGE は Natural ビューの一部であり、テーブルの列 FIRSTNAME および AGE の内容を受け取ります。

parameter	個々のパラメータをターゲットフィールドとして指定する場合、パラメータの数とフォーマットは、対応する選択（上記）に指定された *columns* または *scalar-expressions*（あるいは両方）の数とフォーマットに一致する必要があります（詳細については、「スカラー式」を参照）。

	例： `DEFINE DATA LOCAL` `01 #NAME   (A20)` `01 #AGE    (I2)` `END-DEFINE` `...` `SELECT NAME, AGE` `  INTO #NAME, #AGE` `  FROM SQL-PERSONNEL` `...` ターゲットフィールド #NAME および #AGE は Natural プログラム変数であり、テーブルの列 NAME および AGE の内容を受け取ります。
correlation-name	複数のテーブルが連結（ジョイン）される SELECT * 構造で VIEW 節を使用する場合、指定したビューに含まれるフィールドが複数のテーブルに存在する列を参照するのであれば、*correlation-names*が必要となります。選択する列を識別するために、選択リスト生成時には、これらの列がすべて指定した *correlation-name* で修飾されます。ビューに割り当てる *correlation-name* は、連結（ジョイン）されるテーブルを修飾するために使用される *correlation-names* の1つに対応している必要があります。「ジョインクエリ」も参照してください。 例： `DEFINE DATA LOCAL` `01 PERS VIEW OF SQL-PERSONNEL` `  02 NAME` `  02 FIRST-NAME` `  02 AGE` `END-DEFINE` `...` `SELECT *` `  INTO VIEW PERS A` `  FROM SQL-PERSONNEL A, SQL-PERSONNEL B` `...`

table-expression

table-expression は、FROM 節とオプションの WHERE 節から成ります。 GROUP BY および HAVING 節は指定できません。

例 1：

```
DEFINE DATA LOCAL
01 #NAME      (A20)
01 #FIRSTNAME (A15)
01 #AGE       (I2)
...
END-DEFINE
...
SELECT NAME, FIRSTNAME, AGE
  INTO #NAME, #FIRSTNAME, #AGE
  FROM SQL-PERSONNEL
    WHERE NAME IS NOT NULL
      AND AGE > 20
...
  DISPLAY #NAME #FIRSTNAME #AGE
END-SELECT
...
END
```

例 2：

```
DEFINE DATA LOCAL
01 #COUNT    (I4)
...
END-DEFINE
...
SELECT SINGLE COUNT(*) INTO #COUNT FROM SQL-PERSONNEL
...
```

詳細については、*selection* および *table-expression* を参照してください。

UNION を伴うクエリ

Note: 以降において、「SELECT ステートメント」 は、UNION 演算と連結した複数の選択式で構成されるクエリ式全体の同義語として使用されます。

UNION は、複数の *select-expressions* の結果を結合します。 各 *select-expressions* に指定された列は UNION 互換（数、タイプ、フォーマットが一致）である必要があります。

UNION 演算子に ALL 修飾子が明示的に指定されていないと、重複した余分な行は常に UNION の結果から取り除かれます。 ただし、UNION では、ALL に代わって明示的に指定できる DISTINCT オプションはありません。

例：

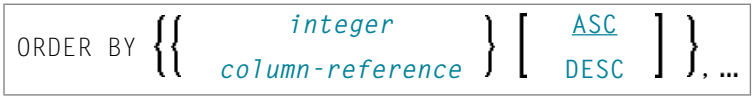
```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
  02 ADDRESS (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE > 55
UNION ALL
SELECT NAME, AGE, ADDRESS
  FROM SQL-EMPLOYEES
  WHERE PERSNR < 100
ORDER BY NAME
...
END-SELECT
...
```

一般的に、任意の数の *select-expressions* を UNION で連結できます。

INTO 節は、最初の *select-expression* にのみ指定する必要があります。

ORDER BY 節は、最後の *select-expression* の後に指定する必要があります。並べ替える列は名前でなく番号で識別する必要があります。

ORDER BY 節



ORDER BY 節は、SELECT ステートメントの結果を特定の順序で整列します。

各ORDER BY節には、結果テーブルの列を指定する必要があります。ほとんどのORDER BY節で、*column-reference*（オプションで修飾された列名）、または列番号によって列を識別できます。UNIONを指定したクエリでは、列は列番号で識別する必要があります。列番号は、*selection*内で左から右へ位置づけ、*integer*値で示します。 この機能により、名前を持たない計算列に基づいて結果を整列できるようになります。

例：

```
DEFINE DATA LOCAL
1 #NAME          (A20)
1 #YEARS-TO-WORK (I2)
END-DEFINE
...
SELECT NAME , 65 - AGE
  INTO #NAME, #YEARS-TO-WORK
  FROM SQL-PERSONNEL
  ORDER BY 2
  ...
```

ORDER BY 節に指定する順序は昇順（ASC）または降順（DESC）です。 ASC がデフォルトです。

例：

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
1 NAME
1 AGE
1 ADDRESS   (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE = 55
  ORDER BY NAME DESC
  ...
```

詳細については、*integer* および *column-reference* を参照してください。

IF NO RECORDS FOUND 節

Note: この節は実際には Natural SQL に属するものではありません。これは Natural 機能が SQL ループ処理に対して有効にされたことを表します。

ストラクチャードモード構文

```
IF NO [RECORDS] [FOUND]
{          ENTER       }
{          statement...}
END-NOREC
```

レポーティングモード構文

```
IF NO [RECORDS] [FOUND]
{       ENTER                    }
{       statement                }
{       DO statement... DOEND    }
```

IF NO RECORDS FOUND 節は、先行する SELECT ステートメントで指定した選択条件に一致するレコードがない場合に処理ループを開始するために使用します。

指定した選択条件に一致するレコードがない場合、IF NO RECORDS FOUND 節により、「空」 のレコードを使用して処理ループが1回実行されます。 これが望ましくない場合は、IF NO RECORDS FOUND 節に ESCAPE BOTTOM ステートメントを指定します。

IF NO RECORDS FOUND 節に 1 つ以上のステートメントを指定すると、処理ループに入る直前にそのステートメントが実行されます。 ループに入る前に実行するステートメントがない場合は、キーワード ENTER を使用する必要があります。

Note: SELECT ステートメントの結果セットが単一行の NULL 値で構成される場合、IF NO RECORDS FOUND 節は実行されません。 この状況は、選択リストが列に対する集積関数 SUM、AVG、MIN、MAX の 1 つで単独に構成され、集積関数で使用するセットが空の場合に起こることがあります。 このような方法で集積関数を使用するときは、IF NO RECORDS FOUND 節を使用せずに、対応する空値インジケータフィールドの値をチェックする必要があります。

データベース値

IF NO RECORDS FOUND 節内のステートメントで他の値の割り当てが行われない限り、Natural は現在のループで指定されたファイルを参照するすべてのデータベースフィールドをリセットして空にします。

システム機能の評価

Natural システム機能は、IF NO RECORDS FOUND 節の結果として処理用に作成される空のレコードに対して 1 回評価されます。

ジョインクエリ

ジョインとは複数のテーブルからデータを取得するクエリです。 参照されるテーブルはすべて FROM 節に指定する必要があります。

例：

```
DEFINE DATA LOCAL
1 #NAME     (A20)
1 #MONEY    (I4)
END-DEFINE
...
SELECT NAME, ACCOUNT
  INTO #NAME, #MONEY
  FROM  SQL-PERSONNEL P, SQL-FINANCE F
  WHERE P.PERSNR = F.PERSNR
    AND F.ACCOUNT > 10000
    ...
```

ジョインは常に FROM 節にリストされたテーブルのデカルト積を形成し、このデカルト積から WHERE 節に指定されたジョイン条件を満たさない行をすべて削除します。

テーブル名が長すぎる場合、 *Correlation-names* を使用して書き込みを保存できます。 選択リストに指定された列が、連結（ジョイン）される複数のテーブルに存在するとき、同じ名前のどの列を選択するかを識別するために *Correlation-names* を使用する必要があります。

SELECT - カーソル指向

Natural FIND ステートメントと同様、カーソル指向の SELECT ステートメントは検索条件を基準に 1 つ以上の DB2 テーブルから行（レコード）のセットを選択するために使用します。 データベースループが開始されるため、ループは LOOP（レポーティングモード）または END-SELECT ステートメントで閉じる必要があります。 この構造では、Natural は FIND ステートメントの場合と同じ処理ループを使用します。

また、アプリケーションプログラムからのカーソル管理は必要なく、Naturalによって自動的に処理されます。

以下に参考情報を示します。

- OPTIMIZE FOR *integer* ROWS
- WITH - 分離レベル
- FETCH FIRST
- WITH INSENSITIVE/SENSITIVE

OPTIMIZE FOR *integer* ROWS

```
[OPTIMIZE FOR integer ROWS]
```

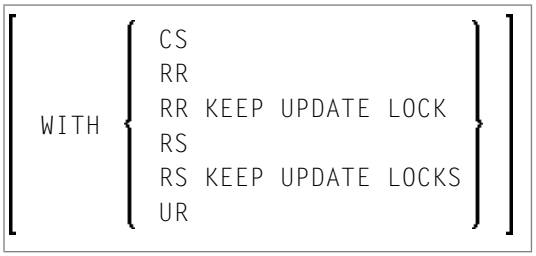
OPTIMIZE FOR *integer* ROWS 節は、結果テーブルから取得する行の数（*integer*）を前もって DB2 に知らせるために使用します。 この節を使用しない場合、DB2 では結果テーブルのすべての行が取得されると仮定し、それに応じて最適化を行います。

このオプションの節は、選択される行の数を把握している場合に役立ちます。実際に選択される行数が *integer* 値（0～2147483647）を超えない場合、*integer* の行を最適化することでパフォーマンスが向上します。

例：

```
SELECT name INTO
#name FROM table WHERE AGE = 2 OPTIMIZE FOR 100 ROWS
```

WITH - 分離レベル



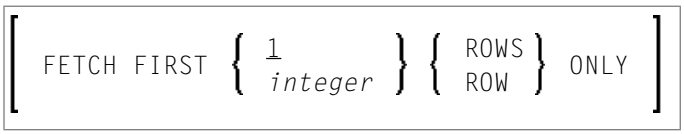
この WITH 節では、ステートメントを実行するときの分離レベルを明示的に指定できます。 次のオプションを使用できます。

オプション	意味
CS	カーソルの安定
RR	繰り返し可能な読み込み
RS	読み込みの安定
RS KEEP UPDATE LOCKS	FOR UPDATE OF 節を指定している場合にのみ有効です。 読み込みの安定と更新ロックの保持です。
RR KEEP UPDATE LOCKS	FOR UPDATE OF 節を指定している場合にのみ有効です。 繰り返し可能な読み込みと更新ロックの保持です。
UR	未コミットの読み込み

WITH UR は、テーブルが読み取り専用のときに SELECT ステートメント内にのみ指定できます。 デフォルトの分離レベルは、ステートメントがバインドされるパッケージまたはプランの分離によって決定されます。 また、デフォルトの分離レベルは、結果テーブルが読み取り専用かどうかによっても変わります。 デフォルトの分離レベルについては、IBM文献を参照してください。

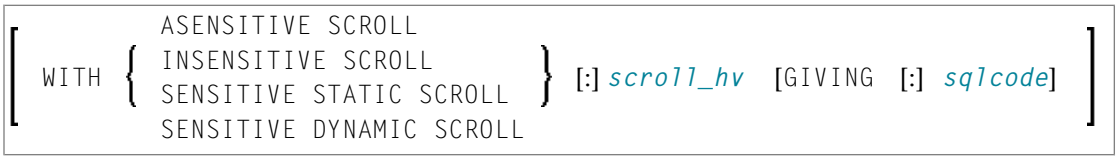
Note: また、このオプションは非カーソル選択にのみ有効です。

FETCH FIRST



FETCH FIRST 節では、フェッチする行の数を制限します。 ごく限られた数の行が必要な場合にこの節を使用すると、結果セットの規模が大きくなる可能性のあるクエリのパフォーマンスが向上します。

WITH INSENSITIVE/SENSITIVE



Natural では、SQL のスクロール可能カーソルを利用できます。そのためには、WITH ASENSITIVE SCROLL 節、WITH SENSITIVE STATIC SCROLL 節、および SENSITVE DYNAMIC SCROLL 節を使用します。 スクロール可能カーソルにより、Natural アプリケーションで結果セット内の任意の行をランダムに位置決めできるようになります。 スクロール不可能なカーソルでは、データは上から下に順番に読み込むことしかできません。

ASENSITIVE スクロール可能カーソルは、INSENSITIVE（カーソルが READ-ONLY の場合）または SENSITIVE DYNAMIC（カーソルが READ-ONLY でない場合）です。

INSENSITIVE および SENSITIVE STATIC スクロール可能カーソルでは一時的な結果テーブルを使用し、DB2 に TEMP データベースが必要になります（IBM の関連 DB2 文献を参照）。

INSENSITIVE SCROLL は、位置決め UPDATE または位置決め DELETE 処理に使用できないカーソルを表します。 また、INSENSITIVE SCROLL カーソルは、一度オープンされると、その後に基本テーブルを対象に実行される UPDATE、DELETE、または INSERT を反映しません。

SENSITIVE STATIC SCROLL は、位置決め UPDATE または位置決め DELETE 処理に使用できるカーソルを表します。 また、SENSITIVE STATIC SCROLL カーソルは、基本テーブルの行の UPDATE、DELETE を反映します。 ただし、INSERT 処理は反映しません。

SENSITIVE DYNAMIC スクロール可能カーソルは、カーソルがオープンされている間、基本テーブルを対象に実行される UPDATE、DELETE、および INSERT を反映します。

以下に参考情報を示します。

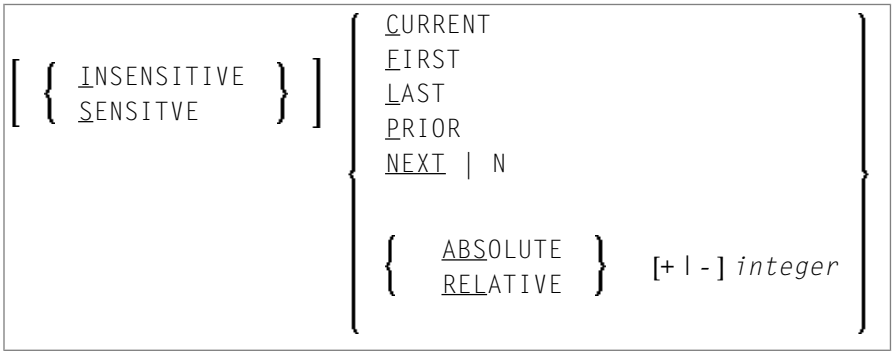
- *scroll_hv*
- *scroll_hv* **- オプション**

- GIVING [:]*sqlcode*

scroll_hv

変数 *scroll_hv* は英数字にする必要があります。

変数 *scroll_hv* では、データベース処理ループの 1 回の実行でフェッチする結果テーブルの行を指定します。 *scroll_hv* の内容は、データベース処理ループサイクルが実行されるたびに評価されます。



scroll_hv **- オプション**

オプション	説明
CURRENT	現在の行を（再度）フェッチします。
FIRST	最初の行をフェッチします。
LAST	最後の行をフェッチします。
NEXT	現在の行の次の行をフェッチします。 これはデフォルト値です。
PRIOR	現在の行の前の行をフェッチします。
+/- integer	ABSOLUTE または RELATIVE との接続でのみ適用されます。 フェッチする行の位置を ABSOLUTE または RELATIVE で指定します。 整数の前にプラス記号（+）またはマイナス記号（-）を入力します。 デフォルト値はプラス（+）です。
ABSOLUTE	*+/- integer* との接続でのみ適用されます。 行をフェッチする結果セット内の絶対位置として *integer* を使用します。
RELATIVE	*+/- integer* との接続でのみ適用されます。 行をフェッチする結果セット内の現在の位置への相対位置として *integer*を使用します。

特別な RDBMS システムにはいくつかの制限があります。

- DB2 ではキーワード CURRENT はサポートされていません。
- SELECT FOR UPDATE ループにおいては、DB2 ではスクロールオプションとして NEXT のみがサポートされています。
- MS SQL Server（ODBC インターフェイス）ではキーワード CURRENT はサポートされていません。
- Adabas D では RELATIVE スクロールはサポートされていません。

GIVING [:]*sqlcode*

GIVING [:] *sqlcode* の指定はオプションです。 指定する場合、Natural 変数 [:] *sqlcode* のフォーマットは I4 にする必要があります。 この変数の値は、基準となる FETCH 処理の DB2 SQLCODE から返されます。 これにより、スクロール可能カーソルがオープンされている間に発生した各種ステータスにアプリケーションで対応できるようになります。 SQLCODE で示される最も重要なステータスコードを次の表に示します。

SQLCODE	説明
0	FETCH 処理は正常に完了しました。データが返されました（ただし、オプション BEFORE または AFTER を指定した FETCH を除く）。
+100	行は見つかりません。カーソルはオープンされたままです。データは返されませんでした。
-1	行に対する FETCH 試行中の一般的なエラーです。

GIVING [:] *sqlcode* を指定する場合、アプリケーションで各種ステータスに対応できる必要があります。 端末 I/O なしで 5 回連続して SQLCODE +100 になった場合、アプリケーションループを回避するために、NDB ランタイムから Natural エラー NAT3296 が発行されます。 アプリケーションでは、ESCAPE ステートメントを実行して処理ループを終了できます。

GIVING [:] *sqlcode* を指定しない場合、SQLCODE 0 および SQLCODE +100 を除き、各 SQLCODE によって Natural エラー NAT3700 が生成され、処理ループが終了します。 SQLCODE +100（行は見つかりません）では処理ループが終了します。

Natural システムライブラリ SYSDB2 内のプログラム例 DEM2SCRL も参照してください。

153 UPDATE - SQL

このchapterでは、次のトピックについて説明します。

機能

SQL UPDATE ステートメントは、カーソルを使用しないでテーブル内の行に対して UPDATE（更新）処理を実行したり（「検索済」 UPDATE）、カーソルが位置づけられた行の列に対して更新処理を実行したり（「位置決め」 UPDATE）するために使用します。

構文説明

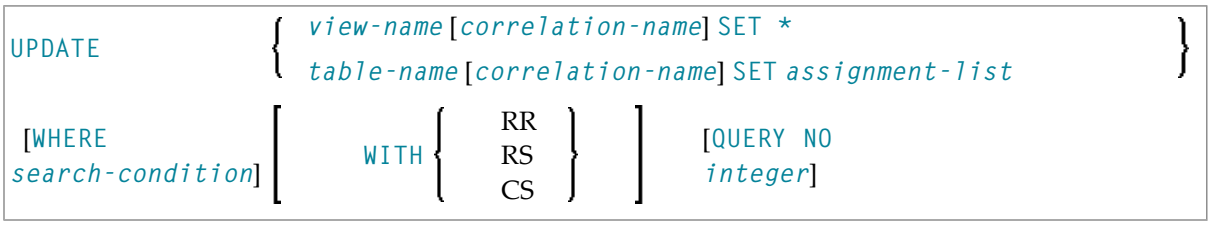
2 つの異なる構造が可能です。

- **構文 1 - 検索済 UPDATE**
- **構文 2 - 位置決め UPDATE**

構文 1 - 検索済 UPDATE

「検索済」 UPDATE ステートメントは、どの SELECT ステートメントにも関連しない独立したステートメントです。単一のステートメントで、テーブルのゼロ、1つ、複数、またはすべての行を更新できます。 更新される行はテーブルに適用する *search-condition* によって決定されます。オプションで、ビュー名およびテーブル名に *correlation-name* を割り当てることができます。

Note: 「検索済」 UPDATE で実際に更新された行の数は、システム変数 *ROWCOUNT（『システム変数』ドキュメントを参照）で確認できます。

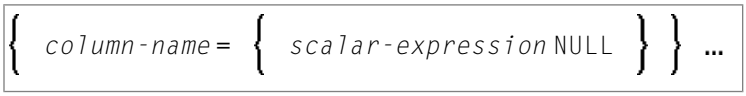


構文図で使用されている記号については、「*構文記号*」を参照してください。

構文要素の説明 - 構文 1：

view-name	ビュー名： DEFINE DATA ステートメントでの定義に従って Natural ビューの名前を参照します。 詳細については、「*基本構文項目*」の *view-name* を参照してください。
SET	**SET** 節： ビューに更新を指定している場合は、ビューのすべての列が更新される必要があるので、SET 節にアスタリスク（*）を指定する必要があります。 テーブルに更新を指定している場合は、SET 節には *assignment-list* または更新する列を含むビューの名前を指定する必要があります。
assignment-list	下記の「*割り当てリスト*」を参照してください。
WHERE *search-condition*	**WHERE** 節： この節では、更新する行の選択条件を指定します。 WHERE 節を指定しないと、テーブル全体が更新されます。

割り当てリスト



assignment-list では、1つ以上の列に値を割り当てることができます。 値は *scalar-expression* または "NULL" のいずれかです。 詳細については、「*スカラー式*」を参照してください。

値 "NULL" が割り当てられている場合は、指定されたフィールドに値（値 "0" または 「空白」でも）は含まれないことを意味します。

SQL 拡張セット

次の構文要素は **SQL 拡張セット**に含まれます。

correlation-name	項目 *correlation-name* は、*table-name* のエイリアス名を表します。 詳細については、「*基本構文項目*」の *correlation-name* を参照してください。
WITH	**WITH** 分離レベル節： この節では、更新する行を検索するときに使用する分離レベルを明示的に指定できます。 DB2 データベースにのみ有効です。 他のデータベースに使用すると、ランタイムエラーが発生します。
QUERYNO *integer*	**QUERYNO** 節： この節は、現在はサポートされていないため無視されます。

構文 2 - 位置決め UPDATE

「位置決め」 UPDATE ステートメントは、常にデータベースループ内のカーソルを参照します。したがって、位置決め UPDATE ステートメントで参照されるテーブルまたはビューは、対応する SELECT ステートメントで参照されるテーブルまたはビューと同一のものである必要があります。そうでない場合はエラーメッセージが返されます。位置決め UPDATE を非カーソル選択で使用することはできません。

一般セットの構文

拡張セットの構文

構文要素の説明 - 構文 2：

view-name	ビュー名： DEFINE DATA ステートメントでの定義に従って Natural ビューの名前を参照します。 詳細については、「*基本構文項目*」の *view-name* を参照してください。
SET * **SET** *assignment-list*	**SET** 節： ビューに更新を指定している場合は、ビューのすべての列が更新される必要があるので、SET 節にアスタリスク（*）を指定する必要があります。 テーブルに更新を指定している場合は、SET 節には *assignment-list* または更新する列を含むビューの名前を指定する必要があります。
WHERE CURRENT OF CURSOR (*r*)	ステートメント参照： (*r*) 表記を使用して、更新する行の選択に使用されたステートメントを参照します。ステートメント参照の指定がない場合、UPDATE ステートメントは、データベースレコードが選択された一番内側のアクティブな処理ループを参照します。
FOR ROW ... OF ROWSET	**FOR ROW ... OF ROWSET** 節： この節は **SQL 拡張セット**に含まれます。 位置決め SQL UPDATE ステートメント用のオプションの FOR ROW ... OF ROWSET 節では、現在の行セットで更新する行を指定します。UPDATE ステートメントが関連する SELECT ステートメントで行セットの位置決めを使用し、その ***INTO*** 節に列の配列が含まれる場合にのみ指定する必要があります。

	この節を指定しない場合、現在の行セットのすべての行が *assignment-list* 内の値で更新されます。 *view-name* SET * を指定している場合、この節は指定できません。

例

例 1 - 検索済 UPDATE

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
2 NAME
2 AGE
...
END-DEFINE
...
ASSIGN AGE = 45
ASSIGN NAME = 'SCHMIDT'
UPDATE PERS SET * WHERE NAME = 'SCHMIDT'
...
```

例 2 - *assignment-list* を使用した検索済 UPDATE

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
2 NAME
2 AGE
...
END-DEFINE
...
```

```
UPDATE SQL-PERSONNEL SET AGE = AGE + 1 WHERE NAME = 'SCHMIDT'
...
```

例 3 - 位置決め UPDATE

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
2 NAME
2 AGE
...
END-DEFINE
...
SELECT * INTO PERS FROM SQL_PERSONNEL WHERE NAME = 'SCHMIDT'
COMPUTE AGE = AGE + 1
UPDATE PERS SET * WHERE CURRENT OF CURSOR
END-SELECT
...
```

例 4 -*assignment-list* を使用した位置決め UPDATE

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
2 NAME
2 AGE
...
END-DEFINE
...
SELECT * INTO PERS FROM SQL-PERSONNEL WHERE NAME = 'SCHMIDT'
UPDATE SQL-PERSONNEL SET AGE = AGE + 1 WHERE CURRENT OF CURSOR
END-SELECT
...
```

154 参照プログラム例

このchapterには、Naturalステートメントおよびシステム変数の参照ドキュメントで参照されている追加のプログラム例が含まれています。 これらすべての例はSYSEXSYNライブラリに含まれています。

ASSIGN

次の例は、ASSIGN/COMPUTE ステートメントの説明で参照されています。

ASGEX1R - ASSIGN（レポーティングモード）

```
** Example 'ASGEX1R': ASSIGN (reporting mode)
************************************************************************
RESET #A (N3)
      #B (A6)
      #C (N0.3)
      #D (N0.5)
      #E (N1.3)
      #F (N5)
      #G (A25)
      #H (A3/1:3)
*
#A = 5                                  WRITE NOTITLE '=' #A
#B = 'ABC'                              WRITE '=' #B
#C = .45                                WRITE '=' #C
#D = #E = -0.12345                      WRITE '=' #D / '=' #E
ASSIGN ROUNDED #F = 199.999             WRITE '=' #F
#G = 'HELLO'                            WRITE '=' #G
*
#H (1) = 'UVW'
#H (3) = 'XYZ'                          WRITE '=' #H (1:3)
*
END
```

プログラム AEDEX1R の出力：

```
#A:    5
#B: ABC
#C:  .450
#D: -.12345
#E: -0.123
#F:    200
```

```
#G: HELLO
#H: UVW     XYZ
```

AT BREAK

次の例は、AT BREAK ステートメントの説明で参照されています。

ATBEX1R - AT BREAK（レポーティングモード）

```
** Example 'ATBEX1R': AT BREAK (reporting mode)
************************************************************************
*
LIMIT 10
READ EMPLOYEES BY CITY
  AT BREAK OF CITY DO
    SKIP 1
  DOEND
  /*
  DISPLAY NOTITLE CITY (IS=ON) COUNTRY (IS=ON) NAME
LOOP
END
```

プログラム ATBEX1R の出力：

```
        CITY          COUNTRY          NAME
-------------------- ------- --------------------

AIKEN                USA     SENKO

AIX EN OTHE          F       GODEFROY

AJACCIO                      CANALE

ALBERTSLUND          DK      PLOUG

ALBUQUERQUE          USA     HAMMOND
                             ROLLING
                             FREEMAN
                             LINCOLN

ALFRETON             UK      GOLDBERG

ALICANTE             E       GOMEZ
```

ATBEX5R - 複数のブレイクレベルがある **AT BREAK** ステートメント（レポーティングモード）

```
** Example 'ATBEX5R': AT BREAK (multiple break levels) (reporting mode)
************************************************************************
RESET LEAVE-DUE-L (N4)
*
LIMIT 5
FIND EMPLOYEES WITH CITY = 'PHILADELPHIA' OR = 'PITTSBURGH'
               SORTED BY CITY DEPT
  MOVE LEAVE-DUE TO LEAVE-DUE-L
  DISPLAY CITY (IS=ON) DEPT (IS=ON) NAME LEAVE-DUE-L
  AT BREAK OF DEPT
    WRITE NOTITLE /
          T*DEPT OLD(DEPT) T*LEAVE-DUE-L SUM(LEAVE-DUE-L) /
  AT BREAK OF CITY
    WRITE NOTITLE
          T*CITY OLD(CITY) T*LEAVE-DUE-L SUM(LEAVE-DUE-L) //
LOOP
*
END
```

プログラム ATBEX5R の出力：

```
        CITY          DEPARTMENT          NAME          LEAVE-DUE-L
                         CODE
-------------------- ---------- -------------------- -----------

PHILADELPHIA         MGMT30     WOLF-TERROINE            11
                                MACKARNESS               27

                     MGMT30                              38

                     TECH10     BUSH                     39
                                NETTLEFOLDS              24

                     TECH10                              63

PHILADELPHIA                                            101


PITTSBURGH           MGMT10     FLETCHER                 34

                     MGMT10                              34
```

```
PITTSBURGH                                                34
```

AT END OF DATA

次の例は、AT END OF DATA ステートメントの説明で参照されています。

AEDEX1R - AT END OF DATA（レポーティングモード）

```
** Example 'AEDEX1R': AT END OF DATA (reporting mode)
************************************************************************
LIMIT 5
EMP. FIND EMPLOYEES WITH CITY = 'STUTTGART'
  IF NO RECORDS FOUND
    ENTER
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
          SALARY (1) CURR-CODE (1)
  /*
  AT END OF DATA DO
    IF *COUNTER (EMP.) = 0 DO
      WRITE 'NO RECORDS FOUND'
      ESCAPE BOTTOM
    DOEND
    WRITE NOTITLE / 'SALARY STATISTICS:'
                  / 7X 'MAXIMUM:' MAX(SALARY(1))  CURR-CODE (1)
                  / 7X 'MINIMUM:' MIN(SALARY(1))  CURR-CODE (1)
                  / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
  DOEND
LOOP
END
```

プログラム AEDEX1R の出力：

```
PERSONNEL         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     CODE
--------- -------------------- -------------------- ---------- --------

11100328  BERGHAUS             ROSE                    70800 DM
11100329  BARTHEL              PETER                   42000 DM
11300313  AECKERLE             SUSANNE                 55200 DM
11300316  KANTE                GABRIELE                61200 DM
11500304  KLUGE                ELKE                    49200 DM

SALARY STATISTICS:
       MAXIMUM:      70800 DM
```

```
       MINIMUM:      42000 DM
       AVERAGE:      55680 DM
```

AT END OF PAGE

次の例は、AT END OF PAGE ステートメントの説明で参照されています。

AEPEX1R - AT END OF PAGE（レポーティングモード）

```
** Example 'AEPEX1R': AT END OF PAGE (reporting mode)
************************************************************************
FORMAT PS=10
LIMIT 10
READ EMPLOYEES BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
          NAME JOB-TITLE 'SALARY' SALARY(1) CURR-CODE (1)
  /*
  AT END OF PAGE DO
     WRITE / 28T 'AVERAGE SALARY: ...' AVER(SALARY(1)) CURR-CODE (1)
  DOEND
  /*
LOOP
END
```

プログラム AEPEX1R の出力：

```
        NAME                  CURRENT             SALARY   CURRENCY
                             POSITION                        CODE
-------------------- ------------------------- ---------- --------

CREMER               ANALYST                        34000 USD
MARKUSH              TRAINEE                        22000 USD
GEE                  MANAGER                        39500 USD
KUNEY                DBA                            40200 USD
NEEDHAM              PROGRAMMER                     32500 USD
JACKSON              PROGRAMMER                     33000 USD
```

```
                              AVERAGE SALARY: ...      33533 USD
```

AT START OF DATA

次の例は、AT START OF DATA ステートメントの説明で参照されています。

ASDEX1R - AT START OF DATA（レポーティングモード）

```
** Example 'ASDEX1R': AT START OF DATA (reporting mode)
************************************************************************
RESET #CITY (A20) #CNTL (A1)
*
REPEAT
  INPUT 'ENTER VALUE FOR CITY' #CITY
  /*
  IF #CITY = ' ' OR= 'END' DO
    STOP
  DOEND
  FIND EMPLOYEES WITH CITY = #CITY
    IF NO RECORDS FOUND DO
      WRITE NOTITLE NOHDR 'NO RECORDS FOUND'
      ESCAPE
    DOEND
    /*
    AT START OF DATA DO
      INPUT (AD=O) 'RECORDS FOUND' *NUMBER //
                   'ENTER ''D'' TO DISPLAY RECORDS' #CNTL (AD=A)
      IF #CNTL NE 'D' DO
        ESCAPE BOTTOM
      DOEND
    DOEND
    /*
    DISPLAY NAME FIRST-NAME
  LOOP
LOOP
END
```

プログラム ASDEX1R の出力：

```
ENTER VALUE FOR CITY PARIS
```

市町村名を入力して確認した後：

```
RECORDS FOUND        26

ENTER 'D' TO DISPLAY RECORDS D
```

「D」を入力して確認した後：

```
        NAME              FIRST-NAME
-------------------- --------------------

MAIZIERE             ELISABETH
MARX                 JEAN-MARIE
REIGNARD             JACQUELINE
RENAUD               MICHEL
REMOUE               GERMAINE
LAVENDA              SALOMON
BROUSSE              GUY
GIORDA               LOUIS
SIECA                FRANCOIS
CENSIER              BERNARD
DUC                  JEAN-PAUL
CAHN                 RAYMOND
MAZUY                ROBERT
FAURIE               HENRI
VALLY                ALAIN
BRETON               JEAN-MARIE
GIGLEUX              JACQUES
KORAB-BRZOZOWSKI     BOGDAN
XOLIN                CHRISTIAN
LEGRIS               ROGER
VVVV
```

AT TOP OF PAGE

次の例は、AT TOP OF PAGE ステートメントの説明で参照されています。

ATPEX1R - AT TOP OF PAGE（レポーティングモード）

```
** Example 'ATPEX1R': AT TOP OF PAGE (reporting mode)
************************************************************************
*
FORMAT PS=15
LIMIT 15
*
READ EMPLOYEES BY NAME STARTING FROM 'L'
  DISPLAY 2X NAME 4X FIRST-NAME CITY DEPT
  WRITE TITLE UNDERLINED 'EMPLOYEE REPORT'
  WRITE TRAILER '-' (78)
  /*
  AT TOP OF PAGE DO
    WRITE 'BEGINNING NAME:' NAME
  DOEND
  /*
  AT END OF PAGE DO
    SKIP 1
    WRITE 'ENDING NAME:   ' NAME
  DOEND
LOOP
END
```

DEFINE SUBROUTINE

次の例は、`DEFINE SUBROUTINE` ステートメントの説明で参照されています。

DSREX1R - DEFINE SUBROUTINE（レポーティングモード）

```
** Example 'DSREX1R': DEFINE SUBROUTINE (reporting mode)
************************************************************************
RESET #ARRAY-ALL (A300)
      #X (N2) #Y (N2)
REDEFINE #ARRAY-ALL (#ARRAY (A75/1:4))
         #ARRAY-ALL (#ALINE (A25/1:4,1:3))
*
FORMAT PS=20
LIMIT 5
*
MOVE 1 TO #X #Y
*
FIND EMPLOYEES WITH NAME = 'SMITH'
  OBTAIN ADDRESS-LINE (1:2)
  /*
  MOVE NAME            TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE           TO #ALINE (#X+3,#Y)
```

```
  IF #Y = 3 DO
    MOVE 1 TO #Y
    PERFORM PRINT
  DOEND
  ELSE DO
    ADD 1 TO #Y
  DOEND
  AT END OF DATA DO
    PERFORM PRINT
  DOEND
LOOP
*
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
RETURN
*
END
```

プログラム AEDEX1R の出力：

```

SMITH                     SMITH                     SMITH
ENGLANDSVEJ 222           3152 SHETLAND ROAD        14100 ESWORTHY RD.
                          MILWAUKEE                 MONTERREY
554349                    877-4563                  994-2260

SMITH                     SMITH
5 HAWTHORN                13002 NEW ARDEN COUR
OAK BROOK                 SILVER SPRING
150-9351                  639-8963
```

FIND

次の例は、FIND ステートメントの説明で参照されています。

FNDFIR - FIRST オプションを指定した **FIND** ステートメント（レポーティングモード）

```

** Example 'FNDFIR': FIND FIRST
************************************************************************
*
FIND FIRST EMPLOYEES WITH CITY = 'DERBY'
*
WRITE NOTITLE 'TOTAL RECORDS SELECTED:' *NUMBER
SKIP 2
WRITE '***FIRST PERSON SELECTED***' //
```

```
      'NAME:      ' NAME /
      'DEPARTMENT:' DEPT /
      'JOB TITLE: ' JOB-TITLE
*
END
```

プログラム FNDFIR の出力：

```
TOTAL RECORDS SELECTED:         141


***FIRST PERSON SELECTED***

NAME:       DEAKIN
DEPARTMENT: SALE01
JOB TITLE:  SALES ACCOUNTANT
```

FNDNUM - NUMBER オプションを指定した **FIND** ステートメント（レポーティングモード）

```
** Example 'FNDNUM': FIND NUMBER
************************************************************************
RESET #BIRTH (D)
*
MOVE EDITED '19500101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOYEES WITH CITY = 'MADRID'
                      WHERE BIRTH LT #BIRTH
*
WRITE NOTITLE 'TOTAL RECORDS SELECTED:       ' *NUMBER
            / 'TOTAL BORN BEFORE 1 JAN 1950: ' *COUNTER
*
END
```

プログラム FNDNUM の出力：

```
TOTAL RECORDS SELECTED:                41
TOTAL BORN BEFORE 1 JAN 1950:          16
```

FNDUNQ - UNIQUE オプションを指定した **FIND** ステートメント（レポーティングモード）

```
** Example 'FNDUNQ': FIND UNIQUE
************************************************************************
RESET #NAME (A20)
*
*
INPUT 'ENTER EMPLOYEE NAME: ' #NAME
IF #NAME = ' '
  STOP
*
FIND UNIQUE EMPLOYEES WITH NAME = #NAME
*
DISPLAY NOTITLE NAME FIRST-NAME JOB-TITLE
*
ON ERROR DO
  WRITE 'NAME EITHER NOT UNIQUE OR DOES NOT EXIST'
  FETCH 'FNDUNQ'
DOEND
*
END
```

プログラム FNDUNQ の出力：

```
ENTER EMPLOYEE NAME: HEURTEBISE
```

「HEURTEBISE」という名前を入力して確認した後：

```
        NAME              FIRST-NAME              CURRENT
                                                 POSITION
-------------------- -------------------- -------------------------

HEURTEBISE           MICHEL               CONTROLEUR DE GESTION
```

FOR

次の例は、FOR ステートメントの説明で参照されています。

FOREX1R - FOR（レポーティングモード）

```
** Example 'FOREX1R': FOR (reporting mode)
************************************************************************
RESET #INDEX (I1)
      #ROOT  (N2.7)
*
FOR #INDEX 1 TO 5
  COMPUTE #ROOT = SQRT (#INDEX)
  WRITE NOTITLE '=' #INDEX 3X '=' #ROOT
LOOP
*
SKIP 1
FOR #INDEX 1 TO 5 STEP 2
  COMPUTE #ROOT = SQRT (#INDEX)
  WRITE '=' #INDEX 3X '=' #ROOT
LOOP
*
END
```

プログラム FOREX1R の出力：

```
#INDEX:    1   #ROOT:   1.0000000
#INDEX:    2   #ROOT:   1.4142135
#INDEX:    3   #ROOT:   1.7320508
#INDEX:    4   #ROOT:   2.0000000
#INDEX:    5   #ROOT:   2.2360679

#INDEX:    1   #ROOT:   1.0000000
#INDEX:    3   #ROOT:   1.7320508
#INDEX:    5   #ROOT:   2.2360679
```

HISTOGRAM

次の例は、HISTOGRAM ステートメントの説明で参照されています。

HSTEX1R - HISTOGRAM（レポーティングモード）

```
** Example 'HSTEX1R': HISTOGRAM (reporting mode)
************************************************************************
*
LIMIT 8
HISTOGRAM EMPLOYEES CITY STARTING FROM 'M'
 DISPLAY NOTITLE CITY
         'NUMBER OF/PERSONS' *NUMBER *COUNTER
LOOP
```

```
*
END
```

プログラム **HSTEX1R** の出力：

```
       CITY          NUMBER OF     CNT
                      PERSONS
-------------------- ---------- ----------

MADISON                       3          1
MADRID                       41          2
MAILLY LE CAMP                1          3
MAMERS                        1          4
MANSFIELD                     4          5
MARSEILLE                     2          6
MATLOCK                       1          7
MELBOURNE                     2          8
```

IF

次の例は、IF ステートメントの説明で参照されています。

IFEX1R - IF（レポーティングモード）

```
** Example 'IFEX1R': IF (reporting mode)
************************************************************************
RESET #BIRTH (D)
*
MOVE EDITED '19450101' TO #BIRTH (EM=YYYYMMDD)
SUSPEND IDENTICAL SUPPRESS
LIMIT 20
*
FND. FIND EMPLOYEES WITH CITY = 'FRANKFURT'
                    SORTED BY NAME BIRTH
  IF SALARY (1) LT 40000
    WRITE NOTITLE '*****' NAME 30X 'SALARY LT 40000'
  ELSE DO
    IF BIRTH GT #BIRTH DO
      FIND VEHICLES WITH PERSONNEL-ID = PERSONNEL-ID (FND.)
        DISPLAY (IS=ON) NAME BIRTH (EM=YYYY-MM-DD)
                        SALARY (1) MAKE (AL=8)
      LOOP
    DOEND
  DOEND
```

```
LOOP
END
```

プログラム IFEX1R の出力：

```
        NAME            DATE      ANNUAL     MAKE
                         OF       SALARY
                       BIRTH
-------------------- ---------- ---------- --------

BAECKER              1956-01-05      74400 BMW
***** BECKER                                        SALARY LT 40000
BLOEMER              1979-11-07      45200 FIAT
FALTER               1954-05-23      70800 FORD
***** FALTER                                        SALARY LT 40000
***** GROTHE                                        SALARY LT 40000
***** HEILBROCK                                     SALARY LT 40000
***** HESCHMANN                                     SALARY LT 40000
HUCH                 1952-09-12      67200 MERCEDES
***** KICKSTEIN                                     SALARY LT 40000
***** KLEENE                                        SALARY LT 40000
***** KRAMER                                        SALARY LT 40000
```

PERFORM BREAK PROCESSING

次の例は、`PERFORM BREAK PROCESSING` ステートメントの説明で参照されています。

PBPEX1R - PERFORM BREAK PROCESSING（レポーティングモード）

```
** Example 'PBPEX1R': PERFORM BREAK PROCESSING (reporting mode)
************************************************************************
RESET #LINE (N2) #INDEX (N2)
*
MOVE 1 TO #LINE
FOR #INDEX 1 TO 18
  PERFORM BREAK PROCESSING
  /*
  AT BREAK OF #INDEX /1/ DO
    WRITE NOTITLE / 'PLEASE COMPLETE LINES 1-9 ABOVE' /
    MOVE 1 TO #LINE
  DOEND
  /*
  WRITE NOTITLE '_' (64) '=' #LINE
  ADD 1 TO #LINE
```

```
LOOP
END
```

プログラム PBPEX1R の出力：

```
_____________________________________________________________________ #LINE:   1
_____________________________________________________________________ #LINE:   2
_____________________________________________________________________ #LINE:   3
_____________________________________________________________________ #LINE:   4
_____________________________________________________________________ #LINE:   5
_____________________________________________________________________ #LINE:   6
_____________________________________________________________________ #LINE:   7
_____________________________________________________________________ #LINE:   8
_____________________________________________________________________ #LINE:   9

PLEASE COMPLETE LINES 1-9 ABOVE

_____________________________________________________________________ #LINE:   1
_____________________________________________________________________ #LINE:   2
_____________________________________________________________________ #LINE:   3
_____________________________________________________________________ #LINE:   4
_____________________________________________________________________ #LINE:   5
_____________________________________________________________________ #LINE:   6
_____________________________________________________________________ #LINE:   7
_____________________________________________________________________ #LINE:   8
_____________________________________________________________________ #LINE:   9

PLEASE COMPLETE LINES 1-9 ABOVE
```

READ

次の例は、READ ステートメントの説明で参照されています。

REAEX1R - READ（レポーティングモード）

```
** Example 'REAEX1R': READ (reporting mode)
************************************************************************
LIMIT 3
*
WRITE 'READ IN PHYSICAL SEQUENCE'
READ EMPLOYEES IN PHYSICAL SEQUENCE
  DISPLAY NOTITLE PERSONNEL-ID NAME *ISN *COUNTER
LOOP
*
WRITE / 'READ IN ISN SEQUENCE'
READ EMPLOYEES BY ISN STARTING FROM 1 ENDING AT 3
```

```
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
LOOP
*
WRITE / 'READ IN NAME SEQUENCE'
READ EMPLOYEES BY NAME
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
LOOP
*
WRITE / 'READ IN NAME SEQUENCE STARTING FROM ''M'''
READ EMPLOYEES BY NAME STARTING FROM 'M'
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
LOOP
*
END
```

プログラム REAEX1R の出力：

```
PERSONNEL         NAME            ISN         CNT
   ID
--------- -------------------- ----------- -----------

READ IN PHYSICAL SEQUENCE
50005800  ADAM                          1           1
50005600  MORENO                        2           2
50005500  BLOND                         3           3

READ IN ISN SEQUENCE
50005800  ADAM                          1           1
50005600  MORENO                        2           2
50005500  BLOND                         3           3

READ IN NAME SEQUENCE
60008339  ABELLAN                     478           1
30000231  ACHIESON                    878           2
50005800  ADAM                          1           3

READ IN NAME SEQUENCE STARTING FROM 'M'
30008125  MACDONALD                   923           1
```

```
20028700  MACKARNESS                    765           2
40000045  MADSEN                        508           3
```

REPEAT

次の例は、REPEAT ステートメントの説明で参照されています。

RPTEX1R - REPEAT（レポーティングモード）

```
** Example 'RPTEX1R': REPEAT (reporting mode)
************************************************************************
RESET #PERS-NR (A8)
*
REPEAT
  INPUT 'ENTER A PERSONNEL NUMBER:' #PERS-NR
  IF #PERS-NR = ' '
    ESCAPE BOTTOM
  FIND EMPLOYEES WITH PERSONNEL-ID = #PERS-NR
    IF NO RECORD FOUND
      REINPUT 'NO RECORD FOUND'
    DISPLAY NOTITLE NAME
  LOOP
LOOP
*
END
```

プログラム RPTEX1R の出力：

```
ENTER A PERSONNEL NUMBER:
```

RPTEX2R - WHILE および **UNTIL** オプションを指定した **REPEAT**（レポーティングモード）

```
** Example 'RPTEX2R': REPEAT (with WHILE and UNTIL option)
************************************************************************
RESET #X (I1) #Y (I1)
*
*
REPEAT WHILE #X <= 5
  ADD 1 TO #X
  WRITE NOTITLE '=' #X
LOOP
*
SKIP 3
REPEAT
  ADD 1 TO #Y
```

```
  WRITE '=' #Y
  UNTIL #Y = 6
LOOP
*
END
```

プログラム RPTEX2R の出力：

```
#X:    1
#X:    2
#X:    3
#X:    4
#X:    5
#X:    6



#Y:    1
#Y:    2
#Y:    3
#Y:    4
#Y:    5
#Y:    6
```

SORT

次の例は、SORT ステートメントの説明で参照されています。

SRTEX1R - SORT（レポーティングモード）

```
** Example 'SRTEX1R': SORT (reporting mode)
************************************************************************
RESET #AVG (P11) #TOTAL-TOTAL (P11) #TOTAL-SALARY (P11)
      #AVER-PERCENT (N3.2)
*
LIMIT 3
FIND EMPLOYEES WITH CITY = 'BOSTON'
  OBTAIN SALARY(1:2)
  COMPUTE #TOTAL-SALARY = SALARY (1) + SALARY (2)
  ACCEPT IF #TOTAL-SALARY GT 0
  /*
  SORT BY PERSONNEL-ID USING #TOTAL-SALARY SALARY(*) CURR-CODE
       GIVE AVER(#TOTAL-SALARY)
  /*
  AT START OF DATA DO
    WRITE NOTITLE '*' (40)
```

```
          'AVG CUMULATIVE SALARY:' *AVER (#TOTAL-SALARY) /
    MOVE *AVER (#TOTAL-SALARY) TO #AVG
  DOEND
  COMPUTE ROUNDED #AVER-PERCENT = #TOTAL-SALARY / #AVG * 100
  ADD #TOTAL-SALARY TO #TOTAL-TOTAL
  /*
  DISPLAY NOTITLE PERSONNEL-ID SALARY (1) SALARY (2)
          #TOTAL-SALARY CURR-CODE (1)
          'PERCENT/OF/AVER' #AVER-PERCENT
  AT END OF DATA
    WRITE / '*' (40) 'TOTAL SALARIES PAID: ' #TOTAL-TOTAL
LOOP
*
END
```

プログラム SRTEX1R の出力：

```
PERSONNEL   ANNUAL     ANNUAL   #TOTAL-SALARY CURRENCY PERCENT
   ID       SALARY     SALARY                   CODE     OF
                                                       AVER
--------- ---------- ---------- ------------- -------- -------

**************************************** AVG CUMULATIVE SALARY:        44633

20000100       31000      29400        60400  USD       135.30
20019200       18000      17100        35100  USD        78.60
20020400       20000      18400        38400  USD        86.00

**************************************** TOTAL SALARIES PAID:        133900
```

STORE

次の例は、STORE ステートメントの説明で参照されています。

STOEX1R - STORE（レポーティングモード）

```
** Example 'STOEX1R': STORE  (reporting mode)
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
RESET #PERSONNEL-ID (A8)
      #NAME         (A20)
      #FIRST-NAME   (A15)
      #BIRTH-D      (D)
      #MAR-STAT     (A1)
      #BIRTH        (A8)
```

```
      #CITY         (A20)
      #COUNTRY      (A3)
      #CONF         (A1)
*
REPEAT
  INPUT 'ENTER A PERSONNEL ID AND NAME (OR ''END'' TO END)' //
        'PERSONNEL-ID : ' #PERSONNEL-ID  //
        'NAME         : ' #NAME          /
        'FIRST-NAME   : ' #FIRST-NAME
  /*
  /*  VALIDATE ENTERED DATA
  /*
  IF #PERSONNEL-ID = 'END' OR #NAME = 'END'
    STOP
  IF #NAME = ' '
    REINPUT WITH TEXT 'ENTER A LAST-NAME' MARK 2 AND SOUND ALARM
  IF #FIRST-NAME = ' '
    REINPUT WITH TEXT 'ENTER A FIRST-NAME' MARK 3 AND SOUND ALARM
  /*
  /*  ENSURE PERSON IS NOT ALREADY ON FILE
  /*
  FIND NUMBER EMPLOYEES WITH PERSONNEL-ID =  #PERSONNEL-ID
  IF *NUMBER > 0
    REINPUT 'PERSON WITH SAME PERSONNEL-ID ALREADY EXISTS'
            MARK 1 AND SOUND ALARM
  MOVE 'N' TO #CONF
  /*
  /*  GET FURTHER INFORMATION
  /*
  INPUT
    'ADDITIONAL PERSONNEL DATA'                          ////
    'PERSONNEL-ID             :' #PERSONNEL-ID (AD=IO) /
    'NAME                     :' #NAME         (AD=IO) /
    'FIRST-NAME               :' #FIRST-NAME   (AD=IO) ///
    'MARITAL STATUS           :' #MAR-STAT             /
    'DATE OF BIRTH (YYYYMMDD) :' #BIRTH                /
    'CITY                     :' #CITY                 /
    'COUNTRY (3 CHARACTERS)   :' #COUNTRY              //
    'ADD THIS RECORD (Y/N)    :' #CONF          (AD=M)
  /*
  /*   ENSURE REQUIRED FIELDS CONTAIN VALID DATA
  /*
  IF NOT (#MAR-STAT = 'S'  OR = 'M' OR = 'D' OR = 'W')
    REINPUT TEXT 'ENTER VALID MARITAL STATUS  S=SINGLE ' -
                 'M=MARRIED D=DIVORCED W=WIDOWED' MARK 1
  IF NOT (#BIRTH = MASK(YYYYMMDD) AND #BIRTH = MASK(1582-2699))
    REINPUT TEXT 'ENTER CORRECT DATE' MARK 2
  IF #CITY  = ' '
    REINPUT TEXT 'ENTER A CITY NAME' MARK 3
  IF #COUNTRY = ' '
    REINPUT TEXT 'ENTER A COUNTRY CODE' MARK 4
  IF NOT (#CONF = 'N' OR= 'Y')
```

```
    REINPUT TEXT 'ENTER Y (YES) OR N (NO)' MARK 5
  IF #CONF = 'N'
    ESCAPE TOP
  /*
  /*  ADD THE RECORD
  /*
  MOVE EDITED #BIRTH TO #BIRTH-D (EM=YYYYMMDD)
  /*
  STORE RECORD IN EMPLOYEES
     WITH PERSONNEL-ID = #PERSONNEL-ID
          NAME         = #NAME
          FIRST-NAME   = #FIRST-NAME
          MAR-STAT     = #MAR-STAT
          BIRTH        = #BIRTH-D
          CITY         = #CITY
          COUNTRY      = #COUNTRY
  END OF TRANSACTION
  /*
  WRITE NOTITLE 'RECORD HAS BEEN ADDED'
  /*
LOOP
END
```

UPDATE

次の例は、UPDATE ステートメントの説明で参照されています。

UPDEX1R - UPDATE（レポーティングモード）

```
** Example 'UPDEX1R': UPDATE (reporting mode)
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
RESET #NAME (A20)
*
INPUT 'ENTER A NAME:' #NAME (AD=M)
IF #NAME = ' '
  STOP
*
FIND EMPLOYEES WITH NAME = #NAME
  IF NO RECORDS FOUND
    REINPUT WITH 'NO RECORDS FOUND'  MARK 1
  /*
  INPUT 'NAME:      ' NAME (AD=O) /
        'FIRST NAME:' FIRST-NAME (AD=M) /
        'CITY:      ' CITY (AD=M)
  /*
  UPDATE USING SAME RECORD
```

```
  /*
  END TRANSACTION
  /*
LOOP
*
END
```

プログラム UPDEX1R の出力：

```
ENTER A NAME:
```

システム変数のプログラム例

次の例は、*OCCURRENCE システム変数の説明で参照されています。

OCC1P - システム変数 ***OCCURRENCE**

```
** Example 'OCC1P': *OCCURRENCE
************************************************************************
DEFINE DATA LOCAL
1 #N1 (N7/1:10)
1 #N2 (N7/1:10,1:10)
1 #N3 (N7/1:10,1:10,1:10)
END-DEFINE
*
CALLNAT 'OCC1N' #N1(*) #N2(1:2,1:4) #N3(1:6,1:7,1:8)
*
END
```

プログラム OCC1P で呼び出されるサブプログラム OCC1N：

```
** Example 'OCC1N': *OCCURRENCE (called by OCC1P)
************************************************************************
DEFINE DATA
PARAMETER
1 PARM1 (N7/1:V)
1 PARM2 (N7/1:V,1:V)
1 PARM3 (N7/1:V,1:V,1:V)
LOCAL
1 #OCC2 (I4/1:2)
1 #OCC3 (I4/1:3)
1 #OCC1 (I4)
END-DEFINE
*
MOVE *OCC(PARM1)   TO #OCC1
```

```
MOVE *OCC(PARM2,*) TO #OCC2(*)
MOVE *OCC(PARM3,*) TO #OCC3(*)
*
DISPLAY #OCC1 #OCC2(*) #OCC3(*)
DISPLAY *OCC(PARM1,*) *OCC(PARM2,*) *OCC(PARM3,*)
*
NEWPAGE
*
WRITE NOHDR
      'Occurrences of 1. parameter:' *OCC(PARM1)
    / 'Occurrences of 1. parameter:' *OCC(PARM1,1)
    / 'Occurrences of 1. parameter:' *OCC(PARM1,*)
    / 'Occurrences of 2. parameter:' *OCC(PARM2,1) *OCC(PARM2,2)
    / 'Occurrences of 2. parameter:' *OCC(PARM2,*)
    / 'Occurrences of 3. parameter:' *OCC(PARM3,1) *OCC(PARM3,2)
                                     *OCC(PARM3,3)
    / 'Occurrences of 3. parameter:' *OCC(PARM3,*)
*
END
```

プログラム OCC1P の出力 - ページ 1：

```
Page      1                                                  05-01-18  10:21:30

   #OCC1       #OCC2       #OCC3
----------- ----------- -----------

         10           2           6
                      4           7
                                  8
         10           2           6
                      4           7
                                  8
```

プログラム OCC1P の出力 - ページ 2：

```
Page      2                                                  05-01-18  10:21:30

Occurrences of 1. parameter:          10
Occurrences of 1. parameter:          10
Occurrences of 1. parameter:          10
Occurrences of 2. parameter:           2           4
Occurrences of 2. parameter:           2           4
Occurrences of 3. parameter:           6           7           8
Occurrences of 3. parameter:           6           7           8
```

OCC2P - システム変数 ***OCCURRENCE**

```
** Example 'OCC2P': *OCCURRENCE
************************************************************************
DEFINE DATA LOCAL
1 #N (N7/1:10)
1 #I (I4)
END-DEFINE
*
FOR #I=1 TO 10
  MOVE #I TO #N(#I)
END-FOR
*
WRITE 'Passing ocurrences 1:5'
CALLNAT 'OCC2N' #N(1:5)
*
WRITE 'Passing ocurrences 5:10'
CALLNAT 'OCC2N' #N(5:10)
*
END
```

プログラム OCC2P で呼び出されるサブプログラム OCC2N：

```
** Example 'OCC2N': *OCCURRENCE (called by OCC2P)
************************************************************************
DEFINE DATA
PARAMETER
1 #ARR (N7/1:V)
LOCAL
1 I    (N7)
END-DEFINE
*
FOR I=1 TO *OCC(#ARR)
  DISPLAY #ARR(I)
END-FOR
*
END
```

プログラム OCC2P の出力：

```
Page      1                                                 05-01-18  10:33:03

Passing ocurrences 1:5
       1
       2
       3
       4
       5
Passing ocurrences 5:10
       5
```

```
      6
      7
      8
      9
     10
```

索引

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

X

あ

わ