Natural for Mainframes

プログラミングガイド

バージョン 4.2.5

October 2009

Natural

This document applies to Natural バージョン 4.2.5 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 プログラミングガイド

このガイドは、Natural リファレンスドキュメントを補足するものです。Natural プログラミングのさまざまな面に関する基本情報、および詳細情報を提供しています。 Natural アプリケーションの作成を始める前に、これらの情報を理解しておく必要があります。「ファーストステップ」も参照してください。 このチュートリアルには、Natural プログラミングの基礎を紹介する一連のセッションが含まれています。

	Natural プログラミングモード	レポーティングモードとストラクチャードモードという 2 つの Natural プログラミングモードの違いについて説明します。 アプリケーションの構造がより明確になるため、通常はストラクチャードモードを排他的に使用することをお勧めします。 したがって、このドキュメントのすべての説明と例は、ストラクチャードモードについて述べられています。 レポーティングモード独自の特性は考慮されていません。
	オブジェクトタイプ	アプリケーション内では、効率的なアプリケーション構造が可能となるよう、いくつかのタイプのプログラミングオブジェクトを使用できます。 このドキュメントでは、データエリア、プログラム、サブプログラム、サブルーチン、ヘルプルーチン、マップなど、さまざまなタイプの Natural プログラミングオブジェクトについて説明します。
	フィールドの定義	プログラムで使用するフィールドを定義する方法について説明します。
	データベース内のデータへのアクセス	Adabas データベースおよび Natural でサポートされる Adabas 以外の各種のデータベース内のデータに Natural を使用してアクセスする方法のさまざまな面について説明します。 原則として、Adabas に関して述べられている機能や例は、他のデータベース管理システムにも適用されます。 違いがある場合は、関連するインターフェイスドキュメントおよび『ステートメント』ドキュメントまたは『パラメータリファレンス』で説明しています。
	データ出力制御	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、つまりデータが表示される方法のさまざまな面について説明します。
	プログラミングのその他のポイント	Natural でのプログラミングに関して、他のドキュメントで述べられていない面について説明します。

インターネットおよび **XML** アクセス用のステートメント	インターネットおよびXMLにアクセスするためのステートメントの概要、メインフレーム環境でこれらのステートメントを使用するための全般的な前提条件、および全般的な制限事項について説明します。
アプリケーションユーザーインターフェイスの設計	アプリケーションで使用するためのユーザーインターフェイスを設計するために使用できる Natural のコンポーネントに関する情報を提供します。
NaturalX	オブジェクトベースアプリケーションを開発する方法について説明します。
Natural 予約キーワード	Naturalプログラミング言語で予約されているすべてのキーワードと語のリストを示します。
参照プログラム例	『*プログラミングガイド*』の上記の各セクションには、Naturalプログラムの例が数多く含まれています。これらのプログラム例には、その他の例（主にレポーティングモードに関するもの）へのリンクも提供されており、それがこの別個のセクションに記載されています。 **注意:** 1. 『*プログラミングガイド*』に示されているすべてのプログラム例は、Natural ライブラリ `SYSEXPG` でもソースコード形式で提供されています。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. Naturalステートメントを使用したその他のプログラム例は、Naturalライブラリ `SYSEXSYN` で提供され、『ステートメント』ドキュメントの「*参照プログラム例*」セクションで説明されています。 3. ライブラリ `SYSEXPG` および `SYSEXSYN` が使用可能であるかを Natural 管理者に確認してください。 4. Naturalプログラム例を使用して Adabas データベースにアクセスするには、Natural ニュークリアスパラメータ `OPTIONS` を `TRUNCATION` に設定する必要があります。

注意: Natural アプリケーションプログラミングインターフェイス（API）の詳細については、『ユーティリティ』ドキュメントの「*SYSEXT - Natural アプリケーションプログラミングインターフェイス*」および「*SYSAPI - Natural アドオン製品の API*」を参照してください。

2 Natural プログラミングモード

プログラミングモードの目的

Natural では次の 2 つのプログラミング方法が提供されます。

- レポーティングモード
- ストラクチャードモード

注意: アプリケーションの構造がより明確になるため、通常はストラクチャードモードを排他的に使用することをお勧めします。

レポーティングモード

レポーティングモードが役に立つのは、複雑なデータやプログラミング構成を必要としない、アドホックレポートおよび小さなプログラムを作成する場合のみです。 レポーティングモードでプログラムを作成すると、小さなプログラムが大きくなり、かつ複雑になりやすいので注意してください。

一部の Natural 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ること、また、その他のステートメントはレポーティングモードで使用すると特別な構造になることに注意してください。 レポーティングモードで使用可能なステートメントの概要については、『*ステートメント*』ドキュメントの「*レポーティングモードのステートメント*」を参照してください。

ストラクチャードモード

ストラクチャードモードは、明確で適切に定義されたプログラム構成で複雑なアプリケーションを実装するときに使用します。 ストラクチャードモードの主な利点は、次のとおりです。

- プログラムを構造化して記述する必要があるため、読みやすく、管理しやすくなります。
- プログラムで使用するすべてのフィールドを、レポーティングモードでできるようなプログラム全体に散在させるのではなく、1つの一元的な場所に定義する必要があるため、データの全体的な制御がはるかに容易になります。

ストラクチャードモードでは、実際のプログラムをコーディングする前に、より詳細な計画を立てる必要があります。このため、多くのプログラミングエラーや非効率な作業を回避できます。

ストラクチャードモードで使用可能なステートメントの概要については、『*ステートメント*』ドキュメントの「*機能別ステートメント*」を参照してください。

プログラミングモードの設定と変更

デフォルトのプログラミングモードは、Natural 管理者がプロファイルパラメータ SM で設定します。設定されたモードは、システムコマンド GLOBALS とセッションパラメータ SM を使用して変更できます。

ストラクチャードモード：	GLOBALS SM=ON
レポーティングモード：	GLOBALS SM=OFF

Naturalのプロファイルおよびセッションパラメータ SMの詳細については、『パラメータリファレンス』の「*SM - ストラクチャードモードでのプログラミング*」を参照してください。

プログラミングモードの変更方法の詳細については、『*Natural の使用*』の「*プログラミングモード*」および『パラメータリファレンス』の「*SM- ストラクチャードモードでのプログラミング*」を参照してください。

機能上の違い

レポーティングモードとストラクチャードモード間の機能上の主な違いは、次のとおりです。

- ループおよび機能ブロックを閉じるための構文
- レポーティングモードで処理ループを閉じる
- ストラクチャードモードで処理ループを閉じる
- プログラム内のデータ要素の位置
- データベース参照

注意: これら 2 つのモード間の機能上の違いの詳細については、『ステートメント』ドキュメントを参照してください。 モードが区別されるステートメントごとに、個別の構文図と構文要素を説明しています。 レポーティングモードで使用可能なステートメントの機能概要については、『ステートメント』ドキュメントの「*レポーティングモードのステートメント*」を参照してください。

ループおよび機能ブロックを閉じるための構文

レポーティングモード：	この目的には、(CLOSE) LOOP ステートメントと DO ... DOEND ステートメントを使用できます。 END-DEFINE、END-DECIDE、および END-SUBROUTINE 以外の END-... ステートメントは使用できません。
ストラクチャードモード：	すべてのループや論理構成は、対応する END-... ステートメントで明示的に閉じる必要があります。 したがって、どのループまたは論理構成がどこで終わるかがすぐにわかります。 LOOP ステートメントおよび DO/DOEND ステートメントは使用できません。

次の 2 つの例は、処理ループと論理条件の構成における 2 つのモードの違いを示しています。

レポーティングモードの例：

レポーティングモードの例では、DO ステートメントおよび DOEND ステートメントを使用して、AT END OF DATA 条件に基づくステートメントブロックの開始と終了を示しています。 END ステートメントによって、アクティブな処理ループがすべて閉じます。

```
READ EMPLOYEES BY PERSONNEL-ID
DISPLAY NAME BIRTH
AT END OF DATA
   DO
      SKIP 2
      WRITE / 'LAST SELECTED:' OLD(NAME)
   DOEND
END
```

ストラクチャードモードの例：

ストラクチャードモードの例では、END-ENDDATA ステートメントを使用して AT END OF DATA 条件を閉じ、END-READ ステートメントを使用して READ ループを閉じています。 結果として、プログラムはさらに明確に構造化され、各構成がどこで開始し、どこで終了するかを即座に確認できます。

```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH

END-DEFINE
READ MYVIEW BY PERSONNEL-ID
   DISPLAY NAME BIRTH
   AT END OF DATA
      SKIP 2
```

```
      WRITE / 'LAST SELECTED:' OLD(NAME)
   END-ENDDATA
END-READ
END
```

レポーティングモードで処理ループを閉じる

処理ループを閉じるには、END、LOOP（またはCLOSE LOOP）、およびSORTのいずれかのステートメントを使用できます。

LOOPステートメントを使用すると複数のループを閉じることができ、ENDステートメントを使用するとアクティブなすべてのループを閉じることができます。1つのステートメントで複数のループを閉じることができるのは、ストラクチャードモードと基本的に異なる点です。

SORTステートメントはすべての処理ループを閉じてから、別の処理ループを開始します。

例 **1 - LOOP**：

```
FIND ...
   FIND ...
   ...
   ...
   LOOP      /* closes inner FIND loop
LOOP         /* closes outer FIND loop
...
...
```

例 **2 - END**：

```
FIND ...
   FIND ...
   ...
   ...
END             /* closes all loops and ends processing
```

例 **3 - SORT**：

```
FIND ...
   FIND ...
   ...
   ...
SORT ...        /* closes all loops, initiates loop
...
END             /* closes SORT loop and ends processing
```

ストラクチャードモードで処理ループを閉じる

ストラクチャードモードでは、それぞれの処理ループに対してループを閉じるための特定のステートメントを使用します。END ステートメントで処理ループを閉じることはできません。SORT ステートメントの前に END-ALL ステートメントを指定する必要があり、SORT ループは END-SORT ステートメントで閉じる必要があります。

例 **1 - FIND**：

```
FIND ...
   FIND ...
   ...
   ...
   END-FIND       /* closes inner FIND loop
END-FIND          /* closes outer FIND loop
...
```

例 **2 - READ**：

```
READ ...
   AT END OF DATA
   ...
   END-ENDDATA
   ...
END-READ           /* closes READ loop
...
...
END
```

例 **3 - SORT**：

```
READ ...
   FIND ...
   ...
   ...
END-ALL         /* closes all loops
SORT            /* opens loop
...
...
END-SORT        /* closes SORT loop
END
```

プログラム内のデータ要素の位置

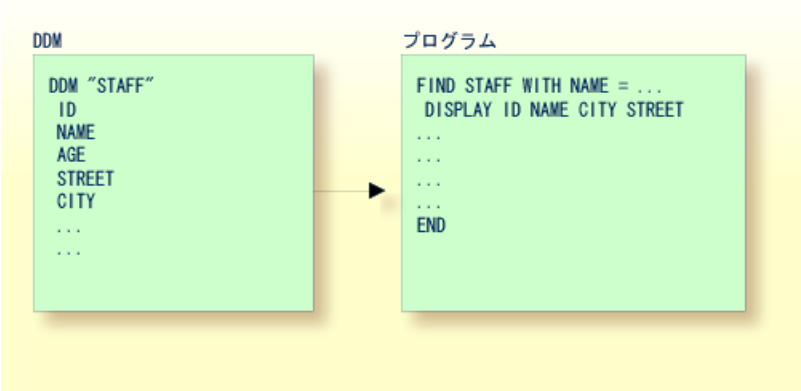
レポーティングモードでは、DEFINE DATAステートメントで定義する必要なくデータベースフィールドを使用することができるうえ、プログラムのどこにでもユーザー定義変数を定義できるので、ユーザー定義変数をプログラム全体に散在させることができます。

ストラクチャードモードでは、使用するすべてのデータ要素を1つの一元的な場所、つまりプログラムの先頭のDEFINE DATAステートメントまたはプログラム外部のデータエリアに定義する必要があります。

データベース参照

レポーティングモード：

レポーティングモードでは、データベースフィールドとDDMは、データエリアにあらかじめ定義しておかなくても参照できます。



ストラクチャードモード：

ストラクチャードモードでは、使用する各データベースフィールドを、「*フィールドの定義*」および「***Adabas** データベースのデータへのアクセス*」で説明されているように、DEFINE DATAステートメントに指定する必要があります。


DDM
DDM "STAFF"
ID
NAME
AGE
STREET
CITY
...
...
プログラム
DEFINE DATA LOCAL
1 VIEWXYZ VIEW OF STAFF
2 ID
2 NAME
2 AGE
2 STREET
2 CITY
END-DEFINE
*
FIND VIEWXYZ WITH NAME = ...
DISPLAY ID NAME CITY STREET
...
...
END-FIND
...
END

3 オブジェクトタイプ

ここでは、効率的なアプリケーション構造を実現するために使用できるNaturalオブジェクトの各種タイプについて説明します。 すべてのNaturalオブジェクトは、Naturalライブラリに保存されています。 Naturalライブラリは、Naturalシステムファイルに含まれています。

- 使用可能なオブジェクトのタイプ
- データエリア
- プログラム、サブプログラム、およびサブルーチン
- リッチ **GUI** ページの処理 **-** アダプタ
- マップ
- ヘルプルーチン
- ソースコードの複数使用 **-** コピーコード
- **Natural** オブジェクトのドキュメント化 **-** テキスト
- コンポーネントベースのアプリケーションの作成 **-** クラス
- **Natural** 以外のファイルの使用 **-** リソース

4 使用可能なオブジェクトのタイプ

プログラミングオブジェクトのタイプ

Naturalアプリケーションでは、いくつかのタイプのオブジェクトを使用して、効率的なアプリケーション構造を実現できます。

Natural オブジェクトのタイプは次のとおりです。

- プログラム
- クラス
- サブプログラム
- アダプタ
- サブルーチン
- コピーコード
- ヘルプルーチン
- テキスト
- マップ
- ローカルデータエリア
- グローバルデータエリア
- パラメータデータエリア

オブジェクトの作成および管理

これらすべてのオブジェクトは、Natural エディタを使用して作成および管理します。

- ローカルデータエリア、グローバルデータエリア、およびパラメータデータエリアは、データエリアエディタで作成および管理します。
- マップは、マップエディタで作成および管理します。
- クラスは、プログラムエディタで作成および管理します。
- 上記以外のすべてのオブジェクトは、プログラムエディタで作成および管理します。

Naturalオブジェクトに適用される命名規則の詳細については、「*オブジェクトの命名規則*」を参照してください。

5 データエリア

データエリアの使用

「*フィールドの定義*」で説明されているように、プログラムで使用するすべてのフィールドは DEFINE DATA ステートメントに定義する必要があります。

フィールドは DEFINE DATA ステートメント内に直接定義できます。または、プログラム外部の別個のデータエリアにフィールドを定義し、DEFINE DATA ステートメントでそのデータエリアを参照できます。

別個のデータエリアとは、複数の Natural プログラム、サブプログラム、サブルーチン、ヘルプルーチン、またはクラスで使用できる Natural オブジェクトを指します。 データエリアには、データ定義モジュール（DDM）から取得されるユーザー定義変数、定数、データベースフィールドなどのデータ要素定義が含まれています。

すべてのデータエリアは、データエリアエディタで作成および編集します。

Natural では、次の 3 つのタイプのデータエリアがサポートされます。

- ローカルデータエリア
- グローバルデータエリア
- パラメータデータエリア

ローカルデータエリア

ローカルとして定義された変数は、単一の Natural プログラミングオブジェクト内でのみ使用できます。 ローカルデータを定義する方法は次の 2 つです。

- プログラム内部に定義します。
- プログラム外部の別個の Natural プログラミングオブジェクト、すなわちローカルデータエリア（LDA）に定義します。

  このようなローカルデータエリアは、そのローカルデータエリアを使用するプログラム、サブプログラム、または外部サブルーチンが実行を開始するときに初期化されます。

アプリケーション構造を明確にして管理しやすくするために、通常はプログラム外部のデータエリアにフィールドを定義する方が便利です。

例 **1**：**DEFINE DATA** ステートメント内に直接定義されたフィールド

次の例では、フィールドはプログラムの DEFINE DATA ステートメント内に直接定義されています。

```
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
1 #VARI-A (A20)
1 #VARI-B (N3.2)
1 #VARI-C (I4)
END-DEFINE
...
```

例 **2**：別のデータエリアに定義されたフィールド

次の例では、同じフィールドがプログラムの DEFINE DATA ステートメントにではなく LDA39 という名前の LDA に定義されており、DEFINE DATA ステートメントにはそのデータエリアへの参照のみ含まれています。

プログラム：

```
DEFINE DATA LOCAL
       USING LDA39
END-DEFINE
  ...
```

ローカルデータエリア LDA39：

```
I T L  Name                            F Length     Miscellaneous
All -- --------------------------------- - ---------- ------------------------->
  V  1 VIEWEMP                                       EMPLOYEES
     2 PERSONNEL-ID                    A          8
     2 FIRST-NAME                      A         20
     2 NAME                            A         20
     1 #VARI-A                         A         20
     1 #VARI-B                         N        3.2
     1 #VARI-C                         I          4
```

グローバルデータエリア

以下では次のトピックについて説明します。

GDA の作成および参照

GDA は、Natural データエリアエディタで作成および変更します。 詳細については、『エディタ』ドキュメントの「データエリアエディタ」を参照してください。

Natural プログラミングオブジェクトで参照される GDA は、この GDA を参照するオブジェクトが保存されているものと同じ Natural ライブラリ、またはこのライブラリに定義されている steplib に保存する必要があります。

注意: 起動における COMMON という名前の GDA の使用：
COMMON という名前の GDA がライブラリに存在する場合は、このライブラリに LOGON すると、ACOMMON という名前のプログラムが自動的に呼び出されます。

重要: 複数の Natural プログラミングオブジェクトが 1 つの GDA を参照するアプリケーションを構築するときは、この GDA 内のデータ要素定義を変更すると、このデータエリアを参照するすべての Natural プログラミングオブジェクトに影響が及ぶことに注意してください。 したがって、このようなオブジェクトは、GDA を変更した後に CATALOG または STOW コマンドを使って再コンパイルする必要があります。

GDA を使用するには、Natural プログラミングオブジェクトで DEFINE DATA ステートメントの GLOBAL 節を使用して GDA を参照する必要があります。 それぞれの Natural プログラミングオブジェクトは1つのグローバルデータエリアのみを参照できます。つまり、DEFINE DATA ステートメントに複数の GLOBAL 節を含めることはできません。

GDA インスタンスの作成および削除

GDA の最初のインスタンスは、これを参照する最初の Natural プログラミングオブジェクトが実行を開始したランタイムに作成され、初期化されます。

GDAインスタンスが作成されると、この中に含まれるデータ値は、このGDAを参照する（DEFINE DATA GLOBAL ステートメント）すべてのNaturalプログラミングオブジェクト、および PERFORM、INPUT、または FETCH ステートメントによって呼び出されるすべての Natural プログラミングオブジェクトで共有できます。GDAインスタンスを共有するすべてのオブジェクトは、同じデータ要素上で機能します。

次の状況が該当する場合に、新しい GDA インスタンスが作成されます。

- いずれかのGDAを参照するサブプログラムがCALLNATステートメントで呼び出された。
- GDAを参照しないサブプログラムが、いずれかのGDAを参照するプログラミングオブジェクトを呼び出した。

GDAの新しいインスタンスが作成されると、現在のGDAインスタンスは中断され、含まれていたデータ値はスタックされます。次にサブプログラムが、新しく作成されたGDAインスタンス内のデータ値を参照します。中断されたGDAインスタンス内のデータ値にアクセスすることはできません。プログラミングオブジェクトは1つのGDAインスタンスしか参照できず、以前のGDAインスタンスにアクセスすることはできません。GDAデータ要素は、CALLNATステートメント内でパラメータとして定義することによって、サブプログラムに渡すことのみが可能です。

サブプログラムが、呼び出し元プログラミングオブジェクトに戻ると、参照されていたGDAインスタンスは削除され、それまで中断していたGDAインスタンスがそのデータ値とともに再開されます。

次のいずれかが該当する場合、GDAインスタンスとその内容は削除されます。

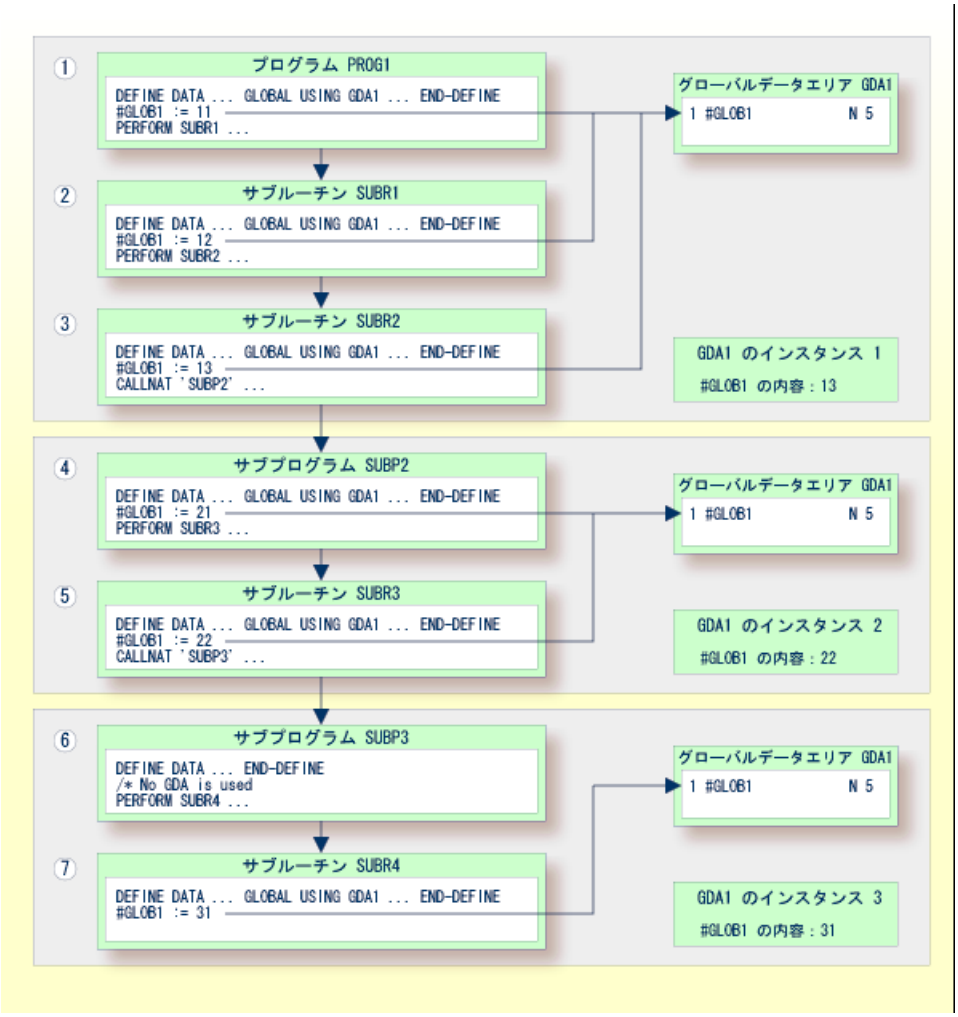
- 次のLOGONが実行された。
- 同じレベルで別のGDAが参照された（レベルについてはこのセクションで後述）。
- RELEASE VARIABLESステートメントが実行された。この場合、GDAインスタンス内の変数は、レベル1のプログラムの実行が終了するか、プログラムがFETCHまたはRUNステートメントで別のプログラムを呼び出したときにリセットされます。

次のセクションの図は、プログラミングオブジェクトがGDAを参照し、GDAインスタンス内のデータ要素を共有する様子を示しています。

GDAインスタンスの共有

次の図は、GDAを参照するサブプログラムが、呼び出し元プログラムが参照するGDAインスタンス内のデータ値を共有できないことを示しています。呼び出し元プログラムと同じGDAを参照するサブプログラムは、このGDAの新しいインスタンスを作成します。ただし、サブプログラムが参照するGDAに定義されているデータ要素は、このサブプログラムが呼び出したサブルーチンやヘルプルーチンで共有できます。

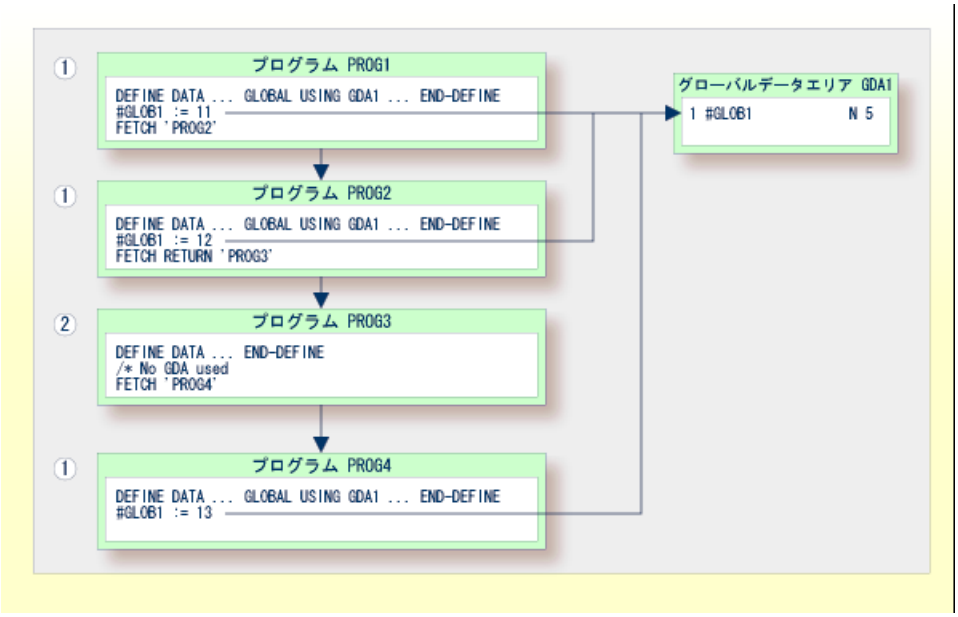
次の図は、GDA1の3つのGDAインスタンスと、データ要素#GLOB1によって各GDAインスタンスに割り当てられる最終値を示しています。数字 ① ～ ⑦ は、プログラミングオブジェクトの階層レベルを示しています。



FETCH または **FETCH RETURN** の使用

次の図は、同じ GDA を参照し、FETCH または FETCH RETURN ステートメントを使用してそれぞれを呼び出す複数のプログラムが、このGDA内に定義されているデータ要素を共有することを示しています。 これらのプログラムのいずれかが GDA を参照しない場合は、以前に参照されていた GDA のインスタンスがアクティブなまま残り、データ要素の値が保持されます。

数字 ① と ② は、プログラミングオブジェクトの階層レベルを示しています。



異なる **GDA** での **FETCH** の使用

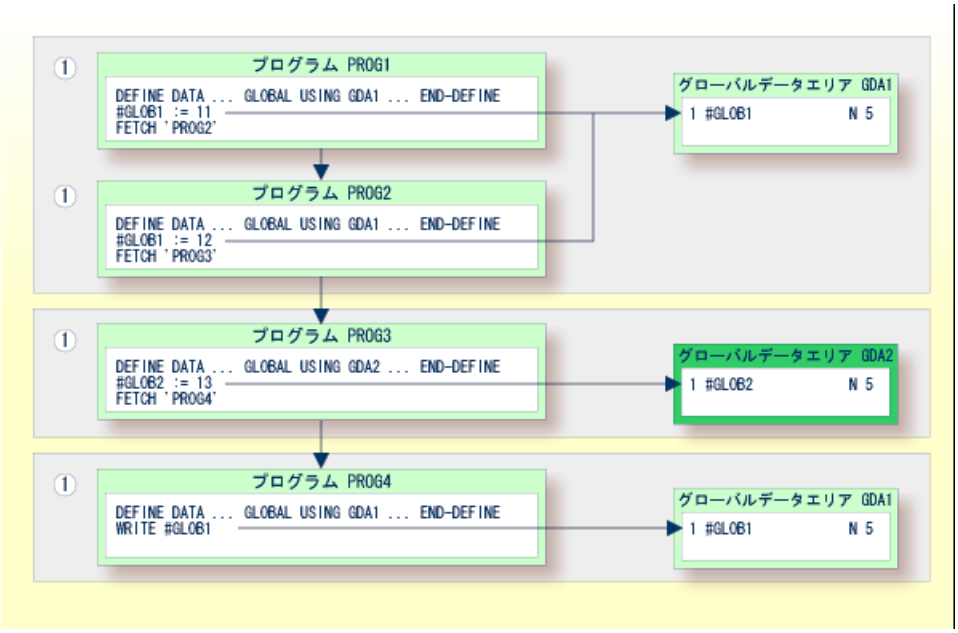
次の図は、プログラムが FETCH ステートメントを使用して、異なる GDA を参照する別のプログラムを呼び出すと、呼び出し元プログラムが参照する GDA（ここでは GDA1）の現在のインスタンスが削除されることを示しています。 その後、この GDA が他のプログラムから再び参照されると、この GDA の新しいインスタンスが作成され、その中のすべてのデータ要素にはそれぞれの初期値が指定されます。

FETCH RETURN ステートメントを使用して、異なる GDA を参照する別のプログラムを呼び出すことはできません。

数字 ① はプログラミングオブジェクトの階層レベルを示しています。

呼び出し元プログラム PROG3 および PROG4 によって、GDA インスタンスは次のような影響を受けます。

- PROG3 内のステートメント GLOBAL USING GDA2 によって、GDA2 のインスタンスが作成され、GDA1 の現在のインスタンスが削除されます。
- PROG4 内のステートメント GLOBAL USING GDA1 によって、GDA2 の現在のインスタンスが削除され、GDA1 の新しいインスタンスが作成されます。 この結果、WRITE ステートメントで値ゼロ（0）が表示されます。



データブロック

データストレージスペースを節約するために、データブロックが含まれる GDA を作成できます。

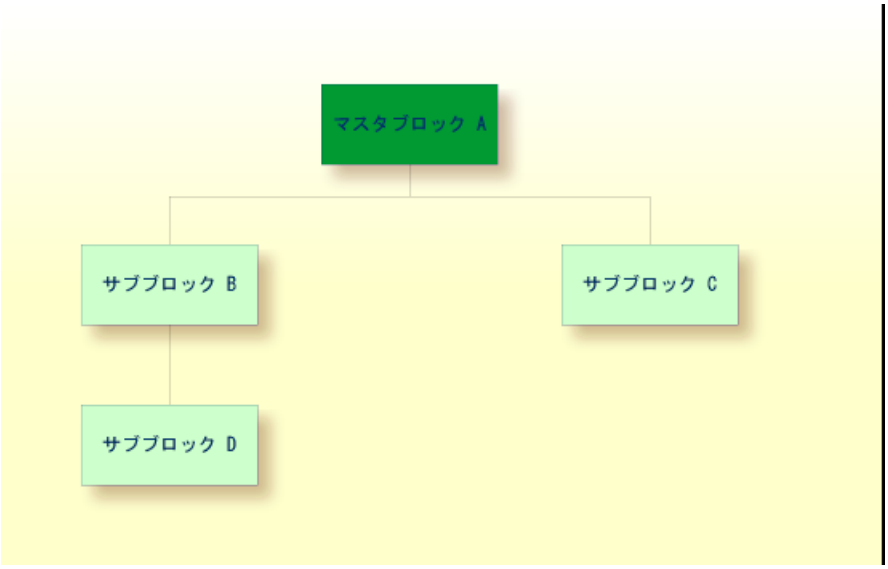
以下では次のトピックについて説明します。

- データブロックの使用例
- データブロックの定義
- ブロック階層

データブロックの使用例

データブロックは、プログラムの実行中に相互に重ねることができるため、ストレージスペースの節約になります。

例えば、次のような階層では、ブロック B とブロック C が同じストレージエリアに割り当てられます。 したがって、ブロック B とブロック C は同時に使用できません。 ブロック B を変更すると、ブロック C の内容が破壊されます。



データブロックの定義

データブロックはデータエリアエディタで定義します。 どのブロックがどのブロックの下位になるかを指定して、ブロック階層を設定します。これは、ブロック定義のコメントフィールドに「親」 ブロックの名前を入力して設定します。

次の例では、SUB-BLOCKB と SUB-BLOCKC が MASTER-BLOCKA の下位になり、SUB-BLOCKD が SUB-BLOCKB の下位になります。

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。

例：

グローバルデータエリア G-BLOCK：

```
I  T  L Name                            F Leng Index/Init/EM/Name/Comment
-  -  - --------------------------------- - ---- ---------------------------------
   B    MASTER-BLOCKA
     1  MB-DATA01                        A   10
   B    SUB-BLOCKB                           MASTER-BLOCKA
     1  SBB-DATA01                       A   20
   B    SUB-BLOCKC                           MASTER-BLOCKA
     1  SBC-DATA01                       A   40
   B    SUB-BLOCKD                           SUB-BLOCKB
     1  SBD-DATA01                       A   40
```

特定のブロックをプログラムで使用可能にするには、DEFINE DATA ステートメント内で次の構文を使用します。

プログラム 1：

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA
END-DEFINE
```

プログラム 2：

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKB
END-DEFINE
```

プログラム 3：

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKC
END-DEFINE
```

プログラム 4：

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKB.SUB-BLOCKD
END-DEFINE
```

この構造では、プログラム1がプログラム2、プログラム3、またはプログラム4と MASTER-BLOCKA 内のデータを共有できます。 ただし、プログラム2とプログラム3は SUB-BLOCKB と SUB-BLOCKC のデータエリアを共有できません。これらのデータブロックは構造と同じレベルにあるため、同じストレージエリアを占有しているからです。

ブロック階層

データブロック階層を使用するときは注意が必要です。 3つのプログラムがデータブロック階層を使用する次のシナリオを想定します。

プログラム 1：

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKB
END-DEFINE
*
MOVE 1234 TO SBB-DATA01
FETCH 'PROGRAM2'
END
```

プログラム 2：

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA
END-DEFINE
*
FETCH 'PROGRAM3'
END
```

プログラム 3：

```
DEFINE DATA GLOBAL
       USING G-BLOCK
       WITH MASTER-BLOCKA.SUB-BLOCKB
END-DEFINE
*
WRITE SBB-DATA01
END
```

説明：

- プログラム 1 は、グローバルデータエリア G-BLOCK を MASTER-BLOCKA および SUB-BLOCKB と共同で使用します。 このプログラムは SUB-BLOCKB 内のフィールドを変更して、プログラム 2 を FETCH しますが、プログラム 2 のデータ定義には MASTER-BLOCKA しか指定されていません。
- プログラム 2 は SUB-BLOCKB をリセット、つまりその内容を削除します。 これは、レベル 1 のプログラム（例えば、FETCH ステートメントで呼び出されたプログラム）は、そのデータ定義に定義されているブロックの下位となるすべてのデータブロックをリセットするからです。
- 次にプログラム 2 はプログラム 3 を FETCH します。プログラム 3 はプログラム 1 で変更されたフィールドを表示するものですが、空の画面を返します。

プログラムレベルの詳細については、「*呼び出されるオブジェクトの複数レベル*」を参照してください。

パラメータデータエリア

サブプログラムは、CALLNAT ステートメントを使用して呼び出されます。CALLNAT ステートメントを使用して、パラメータを呼び出し元オブジェクトからサブプログラムに渡すことができます。

このようなパラメータは、サブプログラム内の DEFINE DATA PARAMETER ステートメントに、次の方法で定義する必要があります。

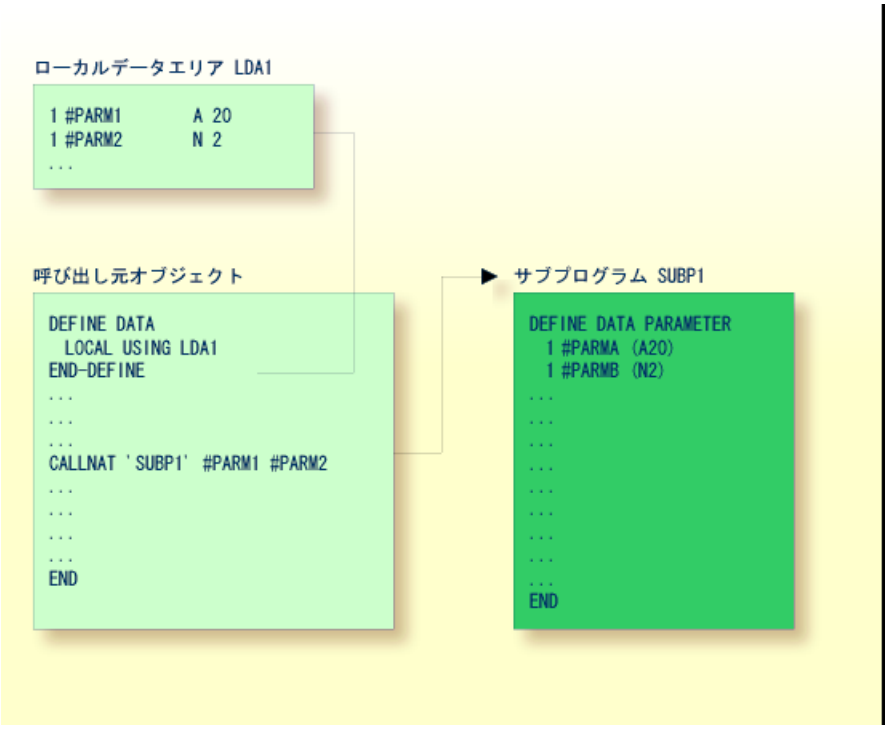
- DEFINE DATA ステートメントの PARAMETER 節内に直接定義します。
- 別のパラメータデータエリア（PDA）に定義して、その PDA を参照する DEFINE DATA PARAMETER ステートメントを指定します。

以下では次のトピックについて説明します。

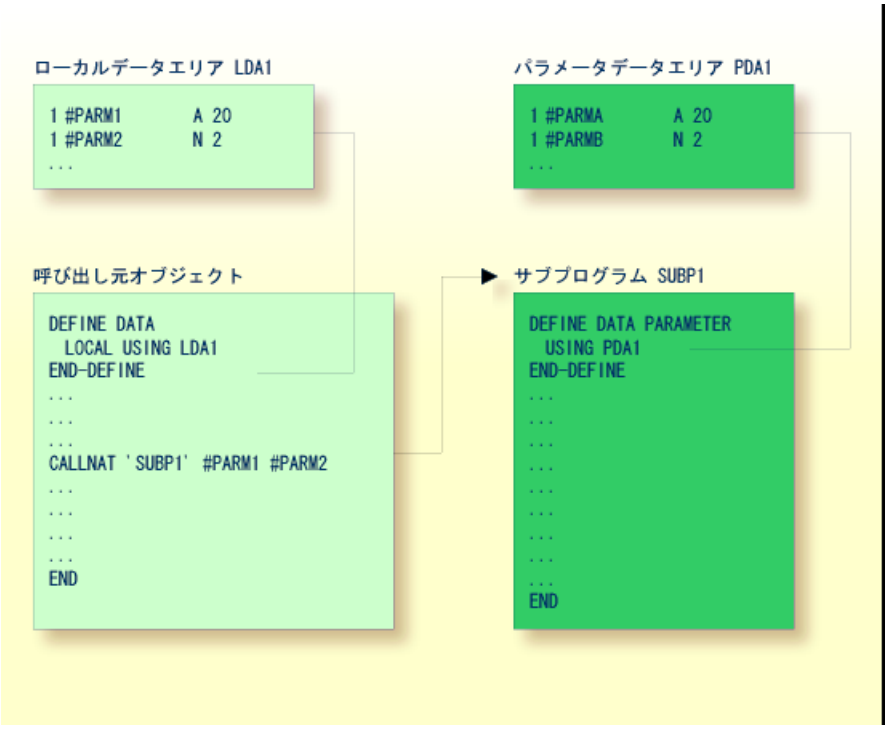
- DEFINE DATA PARAMETER ステートメント内に定義されたパラメータ

- パラメータデータエリアに定義されたパラメータ

DEFINE DATA PARAMETER ステートメント内に定義されたパラメータ



パラメータデータエリアに定義されたパラメータ



同様に、PERFORM ステートメントによって外部サブルーチンに渡されるパラメータは、外部サブルーチン内の DEFINE DATA PARAMETER ステートメントを使用して定義する必要があります。

呼び出し元オブジェクトでは、サブプログラム／サブルーチンに渡されるパラメータ変数をPDA内に定義する必要はありません。上記の図では、呼び出し元オブジェクトが使用する LDA に定義されていますが、GDA に定義することも可能です。

呼び出し元オブジェクト内の CALLNAT/PERFORM ステートメントで指定されるパラメータの順序、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／サブルーチンの DEFINE DATA PARAMETER ステートメントに指定されるフィールドの順序、フォーマット、および長さと正確に一致する必要があります。 ただし、呼び出し元オブジェクト内の変数の名前は、呼び出されるサブプログラム／サブルーチンと同じである必要はありません。パラメータは名前ではなくアドレスで転送されるためです。

呼び出し元プログラムで使用されるデータ要素定義が、サブプログラムまたは外部サブルーチンで使用されるデータ要素定義と同一であることを確実にするために、DEFINE DATA LOCAL USING ステートメントに PDA を指定できます。 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。

6 プログラム、サブプログラム、およびサブルーチン

このドキュメントでは、ルーチン、つまり下位プログラムとして呼び出すことができるオブジェクトタイプについて説明します。

ヘルプルーチンとマップは他のオブジェクトからも呼び出されますが、厳密にはルーチンではないため、個別のドキュメントで説明しています。「ヘルプルーチン」および「マップ」を参照してください。

モジュラーアプリケーション構造

一般に、Naturalアプリケーションは単一の巨大なプログラムで構成されるのではなく、複数のモジュールに分割されます。 これらの各モジュールは、管理可能なサイズの機能ユニットであり、各モジュールは明確に定義された方法でアプリケーションの他のモジュールに接続されています。 これにより、適切に構造化されたアプリケーションが提供されるため、開発およびその後のメンテナンスが大幅に容易かつ迅速に行うことができるようになります。

メインプログラムの実行中には、他のプログラム、サブプログラム、サブルーチン、ヘルプルーチン、およびマップを呼び出すことができます。 続いてこれらのオブジェクトが他のオブジェクトを呼び出すことができます。例えば、サブルーチンが別のサブルーチンを呼び出すこともできます。 したがって、アプリケーションのモジュラー構造は非常に複雑になり、複数のレベルに拡張される可能性があります。

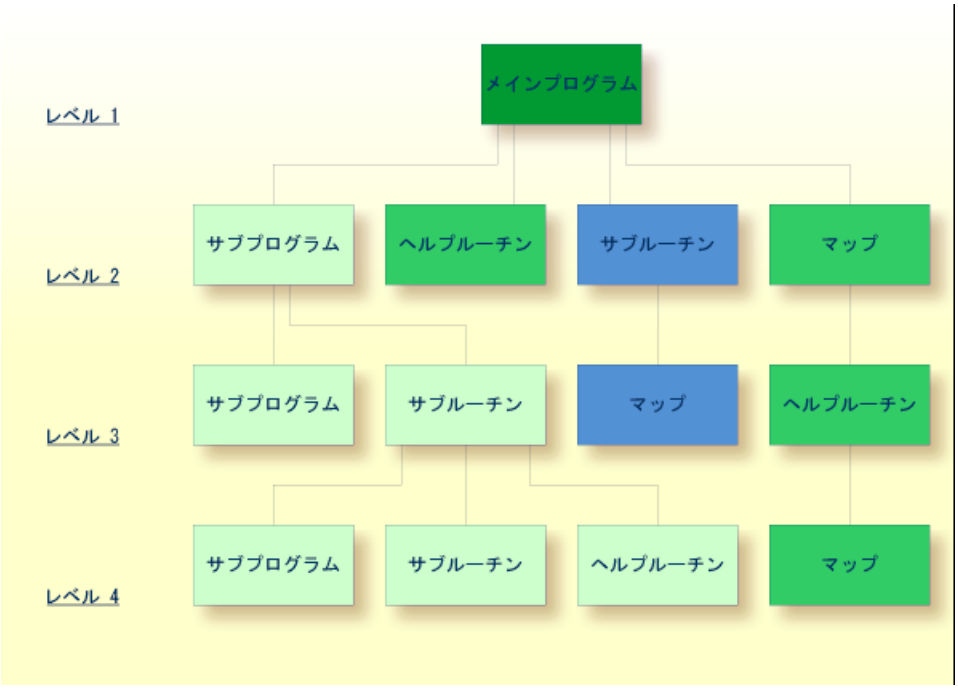
呼び出されるオブジェクトの複数レベル

呼び出される各オブジェクトは、それを呼び出したオブジェクトのレベルの1つ下のレベルになります。つまり、下位オブジェクトを呼び出すたびに、レベル番号は 1 つ増加します。

直接実行されるプログラムはすべてレベル1です。メインプログラムによって直接呼び出されるサブプログラム、サブルーチン、マップ、ヘルプルーチンはレベル2になります。このサブルーチンがサブルーチンを呼び出すと、呼び出されたサブルーチンはレベル 3 になります。

別のオブジェクト内から `FETCH` ステートメントで呼び出されたプログラムは、メインプログラムとして分類され、レベル 1 から動作します。 ただし、`FETCH RETURN` で呼び出されたプログラムは、下位プログラムとして分類され、呼び出し元オブジェクトの1つ下のレベルが割り当てられます。

次の図は、呼び出されるオブジェクトの複数レベルの例と、これらのレベルがどのように数えられるかを示しています。



現在実行中のオブジェクトのレベル番号を確認する場合は、システム変数 *LEVEL を使用できます。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。

このドキュメントでは、ルーチン、つまり下位プログラムとして呼び出すことができる次の Natural オブジェクトタイプについて説明します。

- プログラム
- サブルーチン
- サブプログラム

ヘルプルーチンとマップは他のオブジェクトからも呼び出されますが、厳密にはルーチンではないため、個別のドキュメントで説明しています。「ヘルプルーチン」および「マップ」を参照してください。

基本的に、プログラム、サブプログラムおよびサブルーチンは、データの受け渡し方法やお互いのデータエリアを共有できるかどうかなどがそれぞれ異なります。 したがって、どのオブジェクトタイプをどの目的に使用するかの決定は、アプリケーションのデータ構造に大きく依存します。

プログラム

プログラムは単独で実行し、テストできます。

- ソースプログラムをコンパイルして実行するには、システムコマンド RUN を使用します。
- すでにコンパイルされた形式で存在するプログラムを実行するには、システムコマンド EXECUTE を使用します。

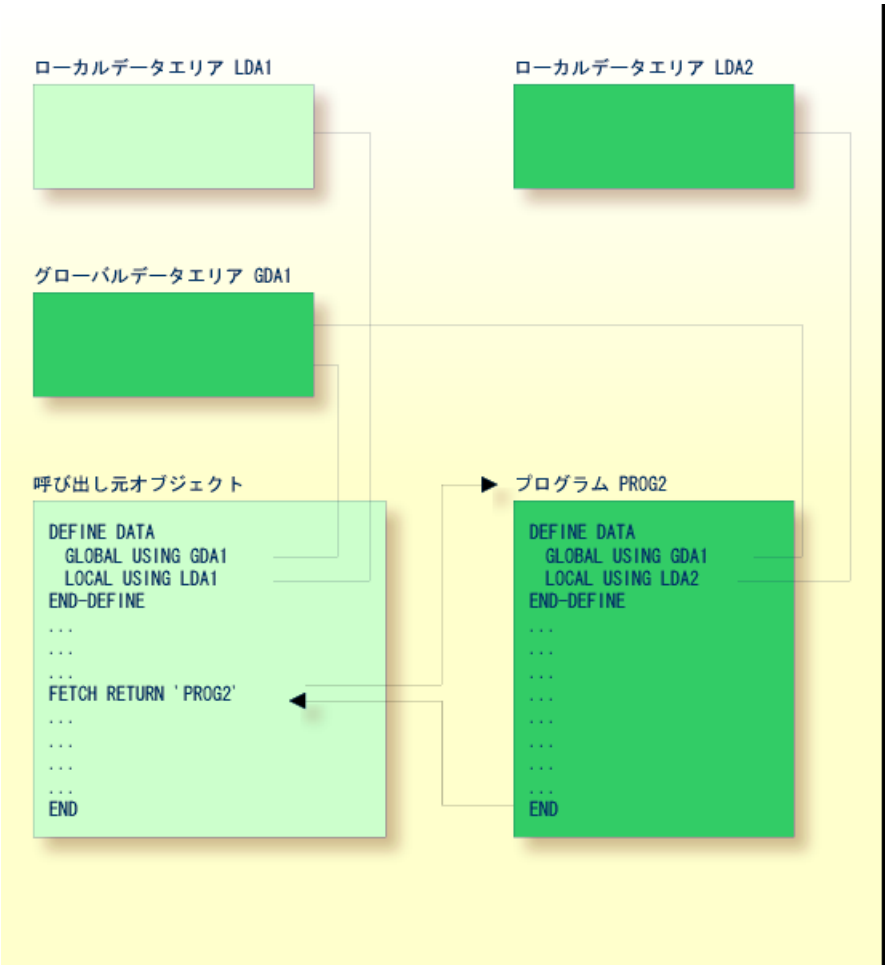
プログラムは、他のオブジェクトから FETCH または FETCH RETURN ステートメントで呼び出すこともできます。呼び出し元オブジェクトは、別のプログラム、サブプログラム、サブルーチン、またはヘルプルーチンのいずれでもかまいません。

- プログラムが FETCH RETURN で呼び出されると、呼び出し元オブジェクトの実行は終了されるのではなく中断され、FETCH されたプログラムは*下位プログラム*としてアクティブになります。 FETCH されたプログラムの実行が終了すると、呼び出し元オブジェクトが再びアクティブになり、その実行は FETCH RETURN ステートメントの次のステートメントから継続します。
- プログラムが FETCH で呼び出されると、呼び出し元オブジェクトの実行は終了し、FETCH されたプログラムがメインプログラムとしてアクティブになります。呼び出し元オブジェクトが、FETCH されたプログラムの終了時に再びアクティブになることはありません。

以下では次のトピックについて説明します。

- FETCH で呼び出されるプログラム

FETCH RETURN で呼び出されるプログラム

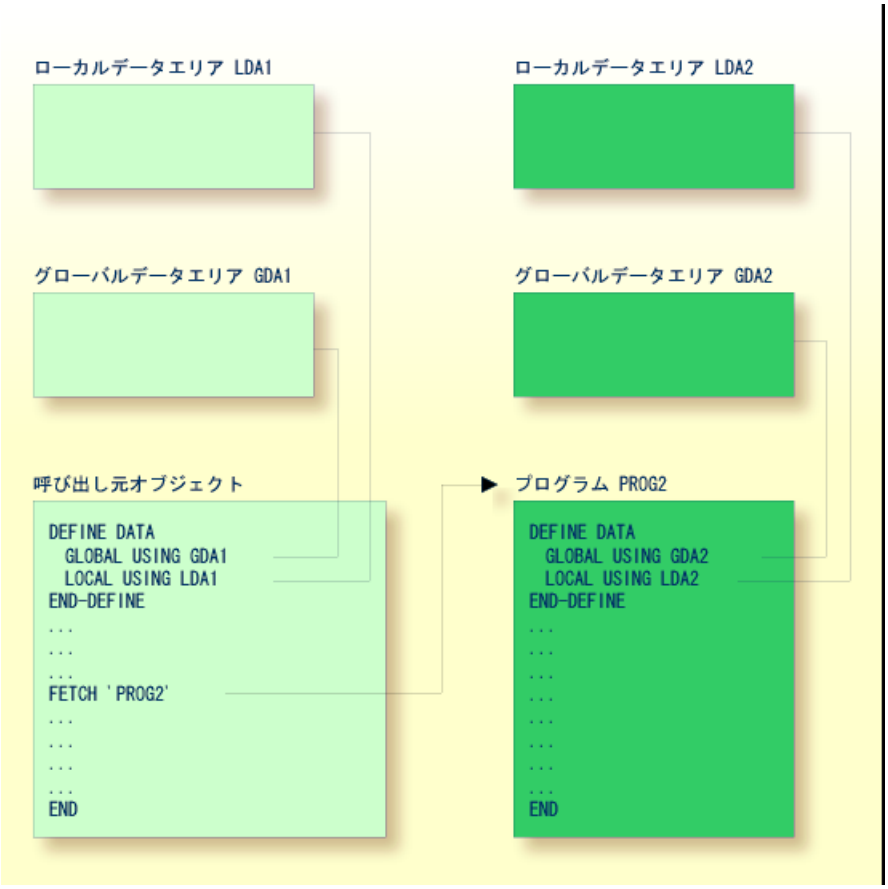


FETCH RETURN で呼び出されるプログラムは、呼び出し元オブジェクトが使用するグローバルデータエリアにアクセスできます。

また、あらゆるプログラムには独自のローカルデータエリアがあり、その中にはそのプログラム内だけで使用されるフィールドが定義されます。

ただし、FETCH RETURN で呼び出されたプログラムが独自のグローバルデータエリアを持つことはできません。

FETCH で呼び出されるプログラム



メインプログラムとして FETCH で呼び出されるプログラムは、上記の図に示したように、通常は独自のグローバルデータエリアを設定します。 ただし、呼び出し元オブジェクトが設定したものと同じグローバルデータエリアを使用することもできます。

注意: ソースプログラムは、RUN ステートメントで呼び出すこともできます。『ステートメント』ドキュメントの RUN ステートメントを参照してください。

サブルーチン

サブルーチンを構成するステートメントは、DEFINE SUBROUTINE ... END-SUBROUTINE ステートメントブロック内に定義する必要があります。

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。

サブルーチンには、*インラインサブルーチン*と*外部サブルーチン*があります。

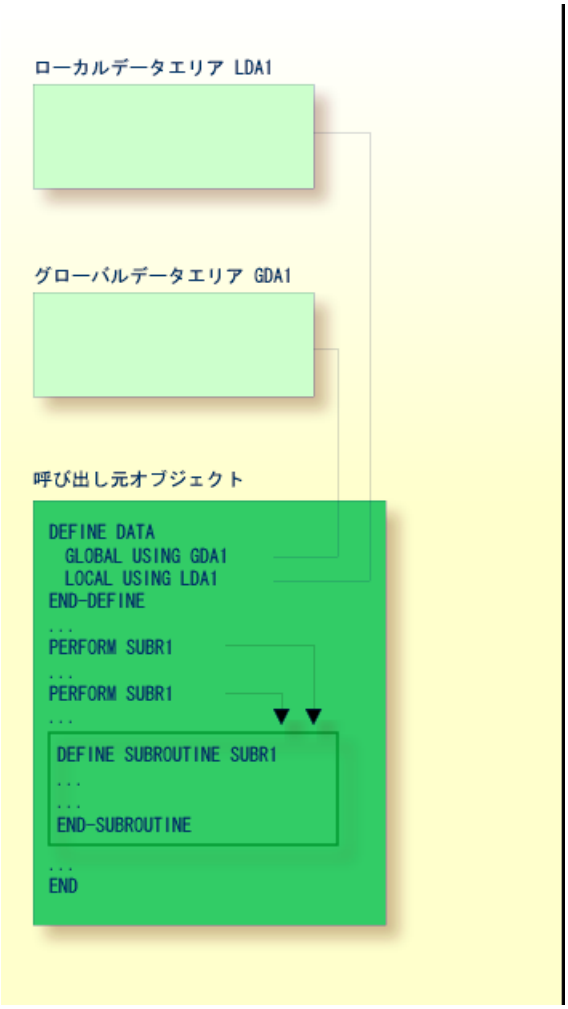
- インラインサブルーチン
  インラインサブルーチンは、このサブルーチンを呼び出す PERFORM ステートメントが含まれるオブジェクト内に定義されるものです。
- 外部サブルーチン
  外部サブルーチンは、このサブルーチンを呼び出すオブジェクトの外部にある別のサブルーチンタイプのオブジェクト内に定義されるものです。

1つのオブジェクト内で繰り返し実行されるコードブロックがある場合は、インラインサブルーチンを使用すると便利です。次に、DEFINE SUBROUTINE ステートメントブロック内でこのブロックを 1 度だけコードし、いくつかの PERFORM ステートメントで呼び出すだけです。

以下では次のトピックについて説明します。

- 外部サブルーチンで使用できるデータ

インラインサブルーチン



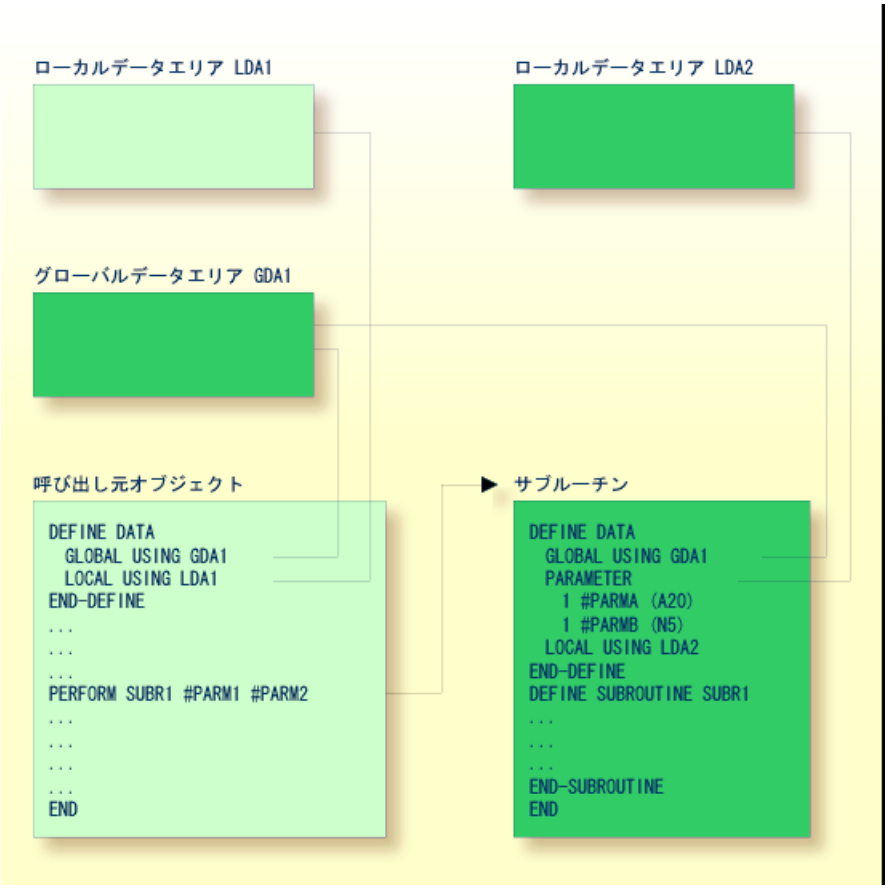
インラインサブルーチンは、タイプがプログラム、サブプログラム、サブルーチン、またはヘルプルーチンのプログラミングオブジェクト内に含めることができます。

インラインサブルーチンが非常に大きく、それを含むオブジェクトの読みやすさが損なわれる場合は、アプリケーションの読みやすさが改善されるように、外部サブルーチンに置き換えることを検討できます。

インラインサブルーチンで使用できるデータ

インラインサブルーチンは、ローカルデータエリアと、このサブルーチンを含むオブジェクトが使用す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、つまりサブルーチンタイプのオブジェクトは単独では実行できません。 これは、他のオブジェクトから呼び出す必要があります。 呼び出し元オブジェクトは、プログラム、サブプログラム、サブルーチン、またはヘルプルーチンのいずれでもかまいません。

外部サブルーチンで使用できるデータ

外部サブルーチンは、呼び出し元オブジェクトが使用するグローバルデータエリアにアクセスできます。

また、PERFORMステートメントを使用して、呼び出し元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ます。これらのパラメータは、サブルーチンのDEFINE DATA PARAMETERステートメント、またはサブルーチンが使用するパラメータデータエリアに定義する必要があります。

また、外部サブルーチンは、そのサブルーチン内のみで使用するフィールドを定義するローカルデータエリアを持つことができます。

ただし、外部サブルーチンが独自のグローバルデータエリアを持つことはできません。

サブプログラム

通常、サブプログラムには、アプリケーション内のさまざまなオブジェクトに使用される一般に使用可能な標準ファンクションが含まれています。

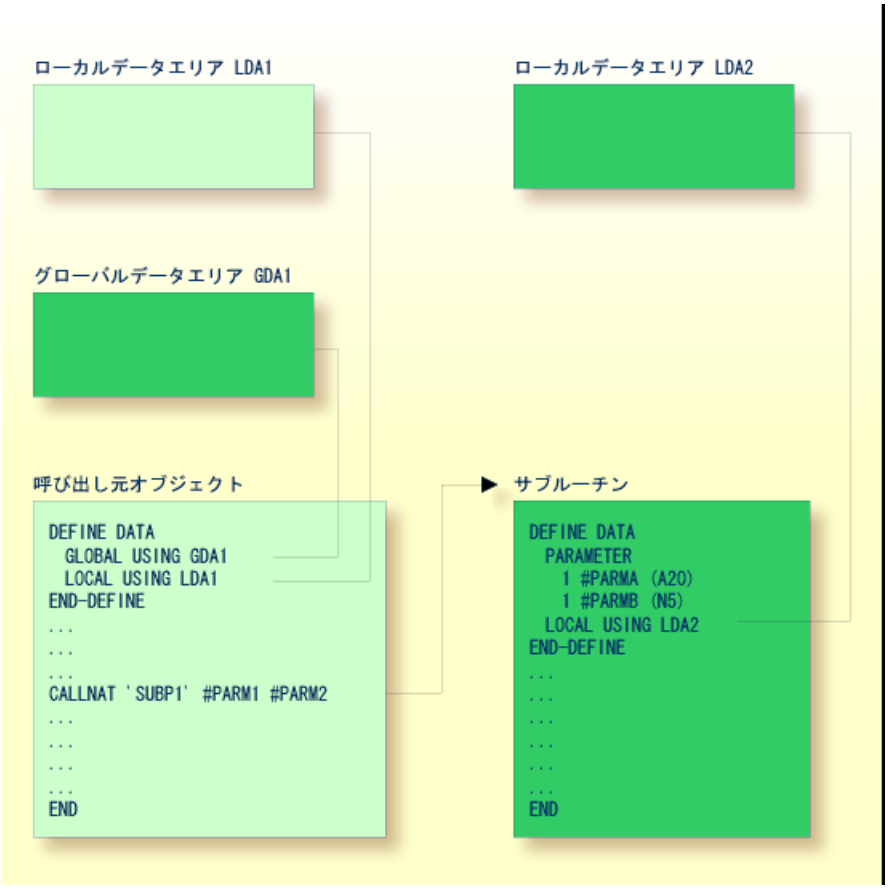
サブプログラムは単独では実行できません。これは、他のオブジェクトから呼び出す必要があります。呼び出し元オブジェクトは、プログラム、サブプログラム、サブルーチン、またはヘルプルーチンのいずれでもかまいません。

サブプログラムは、CALLNATステートメントを使用して呼び出されます。

CALLNATステートメントが実行されると、呼び出し元オブジェクトの実行は中断され、サブプログラムが実行されます。このサブプログラムの実行後は、呼び出し元オブジェクトの実行がCALLNATステートメントの次のステートメントから続行されます。

サブプログラムで使用可能なデータ

CALLNATステートメントを使用して、パラメータを呼び出し元オブジェクトからサブプログラムに渡すことができます。これらのパラメータは、呼び出し元オブジェクトからサブプログラムが使用できる唯一のデータです。これらのパラメータは、サブプログラムのDEFINE DATA PARAMETERステートメント、またはサブプログラムが使用するパラメータデータエリアに定義する必要があります。



また、サブプログラムは、その中で使用するフィールドを定義する独自のローカルデータエリアを持つことができます。

サブプログラムがサブルーチンやヘルプルーチンを呼び出す場合は、そのようなサブルーチンやヘルプルーチンと共有する独自のグローバルデータエリアを設定することもできます。

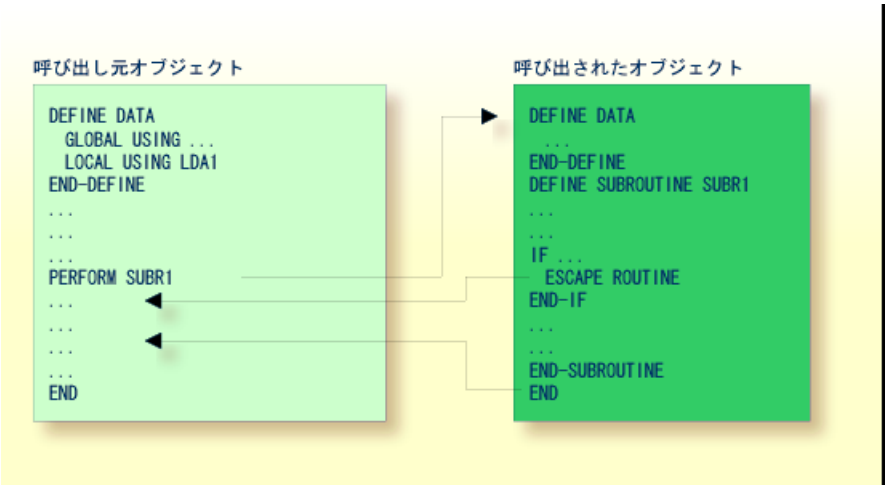
ルーチンを呼び出すときの処理フロー

ルーチン、つまりサブプログラム、外部サブルーチンまたはプログラムをそれぞれ呼び出す CALLNAT、PERFORM、または FETCH RETURN ステートメントが実行されると、呼び出し元オブジェクトの実行は中断され、該当するルーチンの実行が始まります。

ルーチンの実行は、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ルーチンの処理が停止されるまで継続します。

いずれの場合も、呼び出し元オブジェクトの処理は、ルーチンを呼び出すために使用された CALLNAT、PERFORM または FETCH RETURN ステートメントの次のステートメントから続行します。

例：


呼び出し元オブジェクト
DEFINE DATA
GLOBAL USING ...
LOCAL USING LDA1
END-DEFINE
...
...
...
PERFORM SUBR1
...
...
...
...
END
呼び出されたオブジェクト
DEFINE DATA
...
END-DEFINE
DEFINE SUBROUTINE SUBR1
...
...
IF ...
ESCAPE ROUTINE
END-IF
...
...
END-SUBROUTINE
END

7 リッチ GUI ページの処理 - アダプタ

「アダプタ」タイプのNaturalオブジェクトは、Naturalアプリケーション内のリッチGUIページを表現するために使用します。 このオブジェクトタイプは、マップタイプのオブジェクトが端末 I/O 処理に対して行う役割と同様の役割を、リッチ GUI ページの処理に対して行います。 ただし、レイアウト情報を含まない点でマップタイプと異なります。

アダプタタイプのオブジェクトは、外部ページレイアウトから生成されます。 外部で定義および保存されたページレイアウトを使用して、Naturalアプリケーションが表示や修正のために外部 I/O システムにデータを送信できるようにするインターフェイスとして機能します。 アダプタには、このタスクを実行するために必要な Natural コードが含まれています。

アプリケーションプログラムは、`PROCESS PAGE USING` ステートメント内でアダプタを参照します。

オブジェクトタイプ 「アダプタ」 の詳細については、『*Natural for Ajax*』ドキュメントを参照してください。

8 マップ

INPUT ステートメントは、ダイナミックな画面レイアウトの指定の代わりとして、Natural オブジェクトタイプ「マップ」を利用する定義済みマップレイアウトの機能を提供します。

マップ使用の利点

ダイナミック画面レイアウトの指定ではなく定義済みマップレイアウトを使用すると、次のようなさまざまな利点があります。

- プログラムロジックと表示ロジックが結果的に分割されるため、アプリケーションが明確に構造化されます。
- メインプログラムに変更を行うことなくマップレイアウト修正が可能です。
- アプリケーションのユーザーインターフェイスの言語を国際化や地域化に容易に適合させることができます。

マップのようなプログラミングオブジェクトを使用する利点は、既存のNaturalアプリケーションを管理する場合に明白です。

マップのタイプ

マップ（画面レイアウト）は、ユーザーが画面で見ることができるアプリケーションの一部です。

マップには次のタイプがあります。

- 入力マップ
  ユーザーとの対話は入力マップを経由して行われます。
- 出力マップ
  アプリケーションによって出力レポートが生成される場合に、出力マップを使用してこのレポートを画面に表示できます。
- ヘルプマップ
  ヘルプマップは原則的には他のマップに似ていますが、ヘルプとして割り当てられる場合は、ヘルプの目的に使用できることを確実にするために追加チェックが実行されます。

「マップ」オブジェクトタイプは、次の要素で構成されます。

- 画面レイアウトを定義するマップ本文。
- 関連付けられたパラメータデータエリア（PDA）。一種のインターフェイスとして、特定のマップに表示される各フィールドの名前、フォーマット、長さなどのデータ定義が含まれます。

関連トピック：

- 入力フィールドに付加できる選択ボックスの詳細については、『ステートメント』ドキュメントの「INPUT」の「*SB - 選択ボックス*」および『*パラメータリファレンス*』の「*SB - 選択ボックス*」を参照してください。
- 上部を出力マップとして使用し、下部を入力マップとして使用できる画面分割マップの詳細については、『ステートメント』ドキュメントの「INPUT」で「*画面分割機能*」を参照してください。

マップの作成

マップおよびヘルプマップのレイアウトは、マップエディタで作成および編集します。 適切なLDA は、データエリアエディタで作成および管理します。

Naturalがインストールされているプラットフォームに応じて、これらのエディタは、キャラクタユーザーインターフェイスまたはグラフィカルユーザーインターフェイスのいずれかを備えています。

関連トピック：

- データエリアエディタの使用の詳細については、プラットフォーム固有の『*エディタ*』ドキュメントの「*データエリアエディタ*」を参照してください。
- マップエディタの使用の詳細については、プラットフォーム固有の『*エディタ*』ドキュメントの「*マップエディタ*」を参照してください。
- キャラクタユーザーインターフェイスバージョンのNaturalマップエディタによって提供されるあらゆる可能性の包括的な説明については、「*マップエディタのチュートリアル*」を参照してください。
- ダイナミックに指定された画面レイアウトを使用する入力処理の詳細については、『ステートメント』ドキュメントの「INPUT *構文1 - ダイナミック画面レイアウトの指定*」を参照してください。
- マップエディタで作成したマップレイアウトを使用する入力処理の詳細については、『ステートメント』ドキュメントの「INPUT *構文2 - 定義済みマップレイアウトの使用*」を参照してください。

マップ処理の開始／終了

*入力*マップは INPUT USING MAP ステートメントで呼び出されます。

*出力*マップは WRITE USING MAP ステートメントで呼び出されます。

マップの処理は処理ルール内の ESCAPE ROUTINE ステートメントで終了できます。

9 ヘルプルーチン

ヘルプルーチンには、ヘルプ要求の処理を容易にするための特徴があります。 複雑な対話形式のヘルプシステムを実装するために使用できます。 ヘルプルーチンはプログラムエディタで作成します。

ヘルプの呼び出し

Naturalユーザーは、ヘルプ文字をフィールドに入力するか、または、ヘルプキー（通常はPF1）を押してNaturalヘルプルーチンを呼び出すことができます。 デフォルトのヘルプ文字は疑問符（?）です。

- ヘルプ文字は1回しか入力できません。
- ヘルプ文字は入力文字列内で変更される唯一の文字である必要があります。
- ヘルプ文字は入力文字列内の最初の文字である必要があります。

ヘルプルーチンが数値フィールドに対して指定されている場合、Naturalではそのフィールドのヘルプルーチンを呼び出す目的で疑問符を入力できます。 その場合も、Naturalではフィールド入力として有効な数値データが提供されることをチェックします。

まだ指定されていない場合は、`SET KEY` ステートメントでヘルプキーを指定できます。

```
SET KEY PF1=HELP
```

ヘルプルーチンは、それが呼び出されるプログラムまたはマップに指定されている場合にのみ、ユーザーが呼び出すことができます。

ヘルプルーチンの指定

ヘルプルーチンは次のように指定できます。

- プログラム内では、ステートメントレベルおよびフィールドレベルで指定。
- マップ内では、マップレベルおよびフィールドレベルで指定。

ヘルプが指定されていないフィールドに対してヘルプが要求された場合や、参照するフィールドのないヘルプが要求された場合は、ステートメントレベルまたはマップレベルで指定されたヘルプルーチンが呼び出されます。

ヘルプルーチンは、プログラム自体または処理ルール内の `REINPUT USING HELP` ステートメントを使用して呼び出すこともできます。 `REINPUT USING HELP` ステートメントに `MARK` オプションが含まれている場合は、`MARK` された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。 フィールド固有のヘルプルーチンが割り当てられていない場合は、マップのヘルプルーチンが呼び出されます。

ヘルプルーチン内の REINPUT ステートメントは、同じヘルプルーチン内の INPUT ステートメントにのみ適用できます。

ヘルプルーチンの名前は、次に示すような INPUT ステートメントのセッションパラメータ HE で指定できます。

```
INPUT (HE='HELP2112')
```

または、マップエディタの拡張フィールド編集機能を使用して指定することもできます。「*マップの作成*」および『エディタ』ドキュメントを参照してください。

ヘルプルーチンの名前は、英数字定数または名前が含まれる英数字変数で指定できます。 定数の場合は、ヘルプルーチン名をアポストロフィで囲む必要があります。

ヘルプルーチンのプログラミングについて

ヘルプルーチンの処理は、ESCAPE ROUTINE ステートメントで終了できます。

ヘルプルーチンで END OF TRANSACTION または BACKOUT TRANSACTION ステートメントを使用すると、メインプログラムのトランザクションロジックに影響を与えるため注意してください。

ヘルプルーチンとのパラメータの受け渡し

ヘルプルーチンは、現在アクティブなグローバルデータエリアにアクセスできますが、独自のグローバルデータエリアを持つことはできません。 また、独自のローカルデータエリアを持つことができます。

パラメータを使用した、ヘルプルーチンとのデータの受け渡しも可能です。1つのヘルプルーチンに、20 個までの明示的なパラメータと 1 個の暗黙的なパラメータを指定できます。 明示的なパラメータは、ヘルプルーチン名の後に HE オペランドを付けて指定します。

```
HE='MYHELP','001'
```

暗黙的なパラメータとは、ヘルプルーチンが呼び出されたフィールドです。

```
INPUT #A (A5) (HE='YOURHELP','001')
```

上記の 001 は明示的なパラメータであり、#A は暗黙的なパラメータ／フィールドです。

これは、ヘルプルーチンの DEFINE DATA PARAMETER ステートメント内で次のように指定します。

```
DEFINE DATA PARAMETER
1 #PARM1 (A3)          /* explicit parameter
1 #PARM2 (A5)          /* implicit parameter
END-DEFINE
```

暗黙的なパラメータ（上記例の #PARM2）は省略できることに注意してください。 暗黙的なパラメータは、ヘルプが要求されたフィールドにアクセスして、このフィールドにヘルプルーチンからデータを返すために使用します。 例えば、計算機能プログラムをヘルプルーチンとして実装し、計算結果をフィールドに返すことができます。

ヘルプが呼び出されると、データが画面からプログラムデータエリアに渡される前に、ヘルプルーチンが呼び出されます。 これは、ヘルプルーチンは同じ画面トランザクション内で入力されたデータにアクセスできないことを意味しています。

ヘルプ処理が完了すると、画面データはリフレッシュされます。ヘルプルーチンによって変更されたフィールドは更新されます。ヘルプルーチンが呼び出される前にユーザーによって変更されていたフィールドは更新から除外されますが、ヘルプが要求されたフィールドは更新に含まれます。 *例外：*ヘルプが要求されたフィールドが、ダイナミック属性（DY セッションパラメータ）によって複数の部分に分割され、疑問符が入力された部分が、ユーザーによって変更された部分の*後*にある場合、フィールドの内容はヘルプルーチンでは変更されません。

属性制御変数は、ヘルプルーチン処理の後では、たとえヘルプルーチン内で変更されている場合でも再評価されることはありません。

等号オプション

等号（=）は明示的なパラメータとして指定できます。

```
INPUT PERSONNEL-NUMBER (HE='HELPROUT',=)
```

このパラメータは、フィールド名（マップレベルで指定された場合はマップ名）を含む内部フィールド（フォーマット／長さ A65）として処理されます。 対応するヘルプルーチンは、次のように始まります。

```
DEFINE DATA PARAMETER
1 FNAME (A65)              /* contains 'PERSONNEL-NUMBER'
1 FVALUE (N8)              /* value of field (optional)
END-DEFINE
```

このオプションは、フィールド名を読み取り、アプリケーションのオンラインドキュメントまたは Predict データディクショナリにアクセスしてフィールド固有のヘルプを提供する 1 つの共通ヘルプルーチンにアクセスするために使用します。

配列インデックス

ヘルプ文字またはヘルプキーで選択されたフィールドが配列要素である場合、そのインデックスは暗黙的なパラメータ（明示的なパラメータに関係なく、ランクに依存して 1～3）として指定されます。

これらのパラメータのフォーマット／長さは I2 です。

```
INPUT A(*,*)  (HE='HELPROUT',=)
```

対応するヘルプルーチンは、次のように始まります。

```
DEFINE DATA PARAMETER
1 FNAME   (A65)             /* contains 'A'
1 FVALUE  (N8)              /* value of selected element
1 FINDEX1 (I2)              /* 1st dimension index
1 FINDEX2 (I2)              /* 2nd dimension index
END-DEFINE
...
```

ウィンドウとしてのヘルプ

表示するヘルプのサイズを画面サイズよりも小さくすることができます。 この場合、ヘルプは次に示すような、フレームで囲まれたウィンドウとして画面に表示されます。

```
********************************************************************************
                          PERSONNEL INFORMATION
PLEASE ENTER NAME: ?_________________
PLEASE ENTER CITY: _________________
                   +--------------------------+
                   !                          !
                   ! Type in the name of an   !
                   ! employee in the first    !
                   ! field and press ENTER.   !
                   ! You will then receive    !
                   ! a list of all employees  !
                   ! of that name.            !
                   !                          !
                   ! For a list of employees  !
                   ! of a certain name who    !
                   ! live in a certain city,  !
                   ! type in a name in the    !
                   ! first field and a city   !
                   ! in the second field      !
                   ! and press ENTER.         !
*******************!                          !*******************************
                   +--------------------------+
```

ヘルプルーチン内では、ウィンドウのサイズを次の方法で指定できます。

- FORMAT ステートメントを使用します。例えば、FORMAT PS=15 LS=30 のようにページサイズと行サイズを指定します。
- INPUT USING MAP ステートメントを使用します。この場合は、マップのマップ設定で定義されたサイズが使用されます。
- DEFINE WINDOW ステートメントを使用します。この場合は、ウィンドウサイズを明示的に指定することも、内容に応じてNaturalで自動的にウィンドウサイズを決定することもできます。

ヘルプウィンドウの位置は、ヘルプが要求されたフィールドの位置から自動的に計算されます。Naturalでは、対応するフィールドに重ならずにできるだけ近くなる位置にウィンドウを配置します。 DEFINE WINDOW ステートメントを使用すると、自動位置決めを回避して、ユーザー自身がウィンドウ位置を決定できます。

ウィンドウ処理の詳細については、『ステートメント』ドキュメントの「DEFINE WINDOW」ステートメントおよび『*端末コマンド*』ドキュメントの端末コマンド「%W」を参照してください。

10 ソースコードの複数使用 - コピーコード

この章では、コピーコードの利点と使用について説明します。

コピーコードの使用

コピーコードは、INCLUDE ステートメントを使用して別のオブジェクトに組み込むことができるソースコードの一部です。

したがって、複数オブジェクトに同じ形式で現れるステートメントブロックがある場合は、そのステートメントブロックを複数回コーディングする代わりにコピーコードを使用できます。 これにより、コーディングの労力が削減されるうえ、そのブロックが本当に同一であることも保証されます。

コピーコードの処理

コピーコードはコンパイル時に組み込まれます。つまり、コピーコードのソースコード行は、INCLUDE ステートメントを含むオブジェクトに物理的に挿入されるのではなく、コンパイル処理で組み込まれて、結果的に作成されるオブジェクトモジュールの一部となります。

したがって、コピーコードのソースコードを修正するときは、そのコピーコードを使用するすべてオブジェクトも新しくコンパイル（STOW）する必要があります。

注意：

- コピーコードは独自に実行できません。 STOW することはできず、SAVE のみが可能です。
- END ステートメントをコピーコード内に指定することはできません。

詳細については、『ステートメント』ドキュメントの INCLUDE ステートメントを参照してください。

11 Natural オブジェクトのドキュメント化 - テキスト

Natural オブジェクトタイプ 「テキスト」 は、プログラムではなくテキストを記述するために使用します。

テキストオブジェクトの使用

このオブジェクトタイプを使用すると、プログラムのソースコードなどに記述するよりも詳細にNatural オブジェクトをドキュメント化することができます。

テキストオブジェクトは、プログラムをドキュメント化する目的でPredictを使用できない環境でも役立ちます。

テキストの記述

テキストは Natural プログラムエディタを使用して記述します。

プログラムの記述との操作の唯一の違いは、テキストが記述されたまま変わらない点です。つまり、小文字から大文字へ変換されたり、空行が省略されたりすることはありません。ただし、これはエディタプロファイルの空行の省略を`N`に、小文字での編集を`Y`に設定している場合に限ります。詳細については『*エディタ*』ドキュメントの「*エディタのデフォルト*」および「*全般的なデフォルト設定*」を参照してください。詳細については『*エディタ*』ドキュメントを参照してください。

任意のテキストを記述できます。構文チェックはありません。

テキストオブジェクトは `SAVE` のみが可能です。`STOW` することはできません。 エディタに表示されるだけで、`RUN` で実行することはできません。

12 コンポーネントベースのアプリケーションの作成 - クラス

クラスは、オブジェクトベースのプログラミングスタイルを適用するために使用します。 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。

13 Natural 以外のファイルの使用 - リソース

このセクションでは、リソースタイプの Natural オブジェクトについて説明します。

注意: 共有リソースとプライベートリソースを利用できる Natural for Open Systems とは対照的に、Natural for Mainframes では現在共有リソースのみが利用可能です。

リソースとは

リソースとは、HTML ページや GIF のような Natural 外部オブジェクトのことです。Natural アプリケーションからアクセスできるように、システムファイル FNAT または FUSER 上のライブラリに保存されます。

技術的な意味におけるリソースとは、バイナリ形式または文字形式の大きなデータオブジェクトであり、ユーティリティやユーザーアプリケーションの実行への入力やその結果として一時的に処理されるか、永続的に保存されます。

リソースの使用

リソースタイプのオブジェクトは、XML ツールキットによって DTD、XML スキーマ、スタイルシートなどのコンテナとして使用されます。Natural Web インターフェイスでは、GIF や JPEG などのリソースを利用します。 また、リソースタイプのオブジェクトは、XLIFF 変換ファイルの保存に使用できます。

以下では次のトピックについて説明します。

- リソースの命名規則
- リソースの保存

リソースの命名規則

リソースタイプのオブジェクトには、ロングネームとショートネームを付けることができます。

リソースのショートネーム

リソースタイプの各オブジェクトには、8バイトのオブジェクトショートネームが存在します。このショートネームは大文字で表記し、オブジェクトハンドラやユーティリティ INPL および SYSMAIN だけでなく、LIST、DELETE、および RENAME などのシステムコマンドにも指定できます。

リソースのロングネーム

リソースのロングネームは、次の構造を使用して、リソースの3つ目のディレクトリレコードに保存されます。

バイト	フォーマット	内容
1 - 2	B2	行番号 H'0000'
3 - 6	A4	リソースタイプ。通常はリソース名の拡張子。
7	A1	リソースのフォーマット。A は英数字、B はバイナリ、U は Unicode。
8 - 252	A245	リソース名

リソースのロングネームは、システムコマンド LIST を使用して表示できます。ファンクションコード LN を発行すると、*オブジェクトのリスト*に表示されます。

リソースの保存

リソースタイプのオブジェクトは、他の Natural プログラミングオブジェクトソースと同様の方法でライブラリに保存されます。ユーティリティ SYSMAIN および INPL やオブジェクトハンドラを使用して処理できます。

Natural エディタで編集することはできません。

リソースを処理する API

ライブラリ SYSEXT には次のアプリケーションプログラミングインターフェイス（API）が存在し、ユーザーアプリケーションはこれを使用してリソース独自のユーザー出口ルーチンにアクセスできます。

API	目的
USR4208N	ショートネームまたはロングネームを使用して、リソースの書き込み、読み取り、削除を行います。

14 フィールドの定義

このセクションでは、プログラムで使用するフィールドを定義する方法について説明します。これらのフィールドは、データベースフィールドおよびユーザー定義フィールドです。

- DEFINE DATA ステートメントの使用と構造
- ユーザー定義変数
- ダイナミック変数およびフィールドについて
- ダイナミック変数およびラージ変数の使用
- ユーザー定義定数
- 初期値（および RESET ステートメント）
- フィールドの再定義
- 配列
- **X-array**

ここでは、DEFINE DATA ステートメントの主要なオプションのみを説明します。 オプションの詳細については、『ステートメント』ドキュメントを参照してください。

データベースフィールドの詳細については、「*Adabas データベースのデータへのアクセス*」を参照してください。 そのセクションで説明されている Adabas 用の 機能と例は、基本的に他のデータベース管理システムにも適用できます。 システムごとの差異については、『ステートメント』ドキュメントまたは『パラメータリファレンス』ドキュメントの該当するデータベースインターフェイスの説明を参照してください。

15 DEFINE DATA ステートメントの使用と構造

Natural プログラム内にストラクチャードモードで記述されている最初のステートメントは常に、プログラムで使用するフィールドを定義するために使用される DEFINE DATA ステートメントである必要があります。

ソースプログラムのインデントについては、Natural システムコマンド STRUCT の説明を参照してください。

DEFINE DATA ステートメントにおけるフィールド定義

DEFINE DATA ステートメントで、プログラムで使用するすべてのフィールド（ユーザー定義変数およびデータベースフィールド）を定義します。

使用するすべてのフィールドは、DEFINE DATA ステートメントで定義する*必要があります*。

フィールドを定義するには、以下の 2 つの方法があります。

- DEFINE DATA ステートメントそのものでフィールドを定義します（下記参照）。
- プログラム外のローカルデータエリアまたはグローバルデータエリアでフィールドを定義し、DEFINE DATA ステートメントでそのデータエリアを参照します（下記参照）。

複数のプログラム／ルーチンでフィールドを使用する場合は、プログラム外のデータエリアで定義する必要があります。

アプリケーション構造を明確にするために、通常はプログラム外のデータエリアにフィールドを定義することをお勧めします。

データエリアは、データエリアエディタ（『エディタ』ドキュメントを参照）で作成およびメンテナンスします。

以下の最初の例では、プログラムの DEFINE DATA ステートメント内でフィールドを定義しています。 **2** 番目の例では、同じフィールドをローカルデータエリア（LDA）で定義し、DEFINE DATA ステートメントではそのデータエリアへの参照だけを指定しています。

DEFINE DATA ステートメント内でのフィールド定義

以下の例は、DEFINE DATA ステートメント内でのフィールドの定義方法を示しています。

```
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
1 #VARI-A (A20)
1 #VARI-B (N3.2)
1 #VARI-C (I4)
END-DEFINE
...
```

別のデータエリアでのフィールドの定義

以下の例は、ローカルデータエリア（LDA）でのフィールドの定義方法を示しています。

プログラム：

```
DEFINE DATA LOCAL
  USING LDA39
END-DEFINE
...
```

ローカルデータエリア LDA39：

```
I T L Name                            F Leng Index/Init/EM/Name/Comment
- - - --------------------------------- - ---- ---------------------------------
  V 1 VIEWEMP                                 EMPLOYEES
    2 NAME                            A   20
    2 FIRST-NAME                      A   20
    2 PERSONNEL-ID                    A    8
    1 #VARI-A                         A   20
    1 #VARI-B                         N  3.2
    1 #VARI-C                         I    4
```

レベル番号を使用した DEFINE DATA ステートメントの構造化

以下のトピックについて説明します。

定義の構造化およびグループ化

レベル番号は、定義の構造およびグループ化を示すために、DEFINE DATA ステートメント内で使用します。 これは以下に関連します。

- ビュー定義
- フィールドグループ
- 再定義

レベル番号は、01～99 の範囲内の 1 桁または 2 桁の数字です（先頭の 0 は任意）。

一般的に、変数定義はレベル 1 です。

ビュー定義、再定義、およびグループでのレベル番号は、連続している必要があります。レベル番号はスキップできません。

ビュー定義のレベル番号

ビューを定義する場合、ビュー名はレベル 1、ビューを構成するフィールドはレベル 2 で指定する必要があります。 ビュー定義の詳細については、データベースアクセスの説明を参照してください。

ビュー定義のレベル番号の例：

```
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
...
END-DEFINE
```

フィールドグループのレベル番号

グループを定義すると、連続する一連のフィールドを簡単に参照できます。 共通のグループ名の下に複数のフィールドを定義すると、後で、個々のフィールド名の代わりにグループ名のみを指定することによって、プログラム内でフィールドを参照できます。

グループ名はレベル1で指定し、グループに含まれるフィールドは1つ低いレベルにする必要があります。

グループ名には、ユーザー定義変数と同じ命名規則が適用されます。

グループのレベル番号の例：

```
DEFINE DATA LOCAL
1 #FIELDA (N2.2)
1 #FIELDB (I4)
1 #GROUPA
  2 #FIELDC (A20)
  2 #FIELDD (A10)
  2 #FIELDE (N3.2)
1 #FIELDF (A2)
...
END-DEFINE
```

この例では、フィールド #FIELDC、#FIELDD、および #FIELDE は、共通のグループ名 #GROUPA の下に定義されています。 他の 3 つのフィールドはグループの一部ではありません。 #GROUPA はグループ名として機能するだけで、それ自体はフィールドではないことに注意してください（したがって、フォーマット／長さ定義を持っていません）。

再定義のレベル番号

フィールドを再定義する場合、REDEFINE オプションは元のフィールドと同じレベルにする必要があります。再定義から作成するフィールドは、1つ低いレベルにする必要があります。再定義の詳細については、「*フィールドの再定義*」を参照してください。

再定義のレベル番号の例：

```
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF STAFFDDM
  2 BIRTH
  2 REDEFINE BIRTH
    3 #YEAR-OF-BIRTH  (N4)
    3 #MONTH-OF-BIRTH (N2)
    3 #DAY-OF-BIRTH   (N2)
1 #FIELDA (A20)
1 REDEFINE #FIELDA
```

```
  2 #SUBFIELD1 (N5)
  2 #SUBFIELD2 (A10)
  2 #SUBFIELD3 (N5)
...
END-DEFINE
```

この例では、データベースフィールド BIRTH は、3 つのユーザー定義変数として再定義され、ユーザー定義変数の #FIELDA は、他の 3 つのユーザー定義変数として再定義されています。

16 ユーザー定義変数

ユーザー定義変数は、プログラム内でユーザーが定義するフィールドです。 ユーザー定義変数は、追加の処理または表示のためにプログラム処理の特定のポイントで得られた値または中間結果を保存するために使用します。

『*Natural の使用*』ドキュメントの「*ユーザー定義変数の命名規則*」も参照してください。

変数の定義

ユーザー定義変数を定義するには、`DEFINE DATA` ステートメントで名前とフォーマット／長さを指定します。

以下の表記に従って、変数の特徴を定義します。

```
(r,format-length/index)
```

この表記は変数名の後に、任意で 1 つ以上の空白で区切って記述します。

各表記要素の間に空白を入れることはできません。

各要素は、必要に応じて個別に指定できます。まとめて指定する場合は、上記の文字で区切る必要があります。

例：

以下の例では、英数字フォーマットで 10 桁の長さのユーザー定義変数が、`#FIELD1` という名前で定義されています。

```
DEFINE DATA LOCAL
1 #FIELD1 (A10)
...
END-DEFINE
```

注意:

1. ストラクチャードモードで操作している場合、またはプログラムに `DEFINE DATA LOCAL` 節が含まれている場合、ステートメント内で変数をダイナミックに定義することはできません。
2. これは、アプリケーションに依存しない変数（AIV）には適用されません。「*アプリケーションに依存しない変数の定義*」も参照してください。

表記 (r) を使用したデータベースフィールドの参照

ステートメントラベルまたはソースコード行番号は、前のNaturalステートメントを参照するために使用できます。この参照は、Naturalのデフォルトの参照を変更したり（各ステートメントの記述に従って実行可能な場合）、処理内容を説明したりするために使用できます。「*ループ処理*」の「*プログラム内のステートメント参照*」も参照してください。

以下では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のデフォルトの参照
- ステートメントラベルによる参照
- ソースコード行番号による参照

データベースフィールドのデフォルトの参照

ステートメント参照表記を指定しない場合、通常は以下の参照が適用されます。

- デフォルトでは、最も内側のアクティブなデータベースループ（FIND、READ、HISTOGRAM）で読み込まれているフィールドが参照されます。
- アクティブなデータベースループでフィールドが読み込まれていない場合、まだクローズされていないループ内にあり、かつ、当該フィールドが読み込まれている、直前の GET ステートメント（レポーティングモードでは FIND FIRST または FIND UNIQUE ステートメントも可）が参照されます。

ステートメントラベルによる参照

ループ処理を開始したり、データ要素によるデータベースへのアクセスを実行したりするNaturalステートメントには、後で参照するための記号ラベルをマークできます。

ラベルは、参照するオブジェクトの前に *label.* という形式で指定するか、または参照するオブジェクトの後に（*label.*）という形式で指定します。ただし、同時に指定することはできません。

ラベルの命名規則は、変数の名前の命名規則と同一です。 ラベル名の後のピリオドは、そのエントリをラベルとして識別するために機能します。

例：

```
...
 RD. READ PERSON-VIEW BY NAME STARTING FROM 'JONES'
   FD. FIND AUTO-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (FD.)
         DISPLAY NAME (RD.) FIRST-NAME (RD.) MAKE (FD.)
    END-FIND
 END-READ
 ...
```

ソースコード行番号による参照

ステートメントは、ステートメントが記述されているソースコード行を使用して参照することもできます。

4 桁すべての行番号を指定する必要があります（先行ゼロは省略不可能）。

例：

```
...
0110 FIND EMPLOYEES-VIEW WITH NAME = 'SMITH'
0120   FIND VEHICLES-VIEW WITH MODEL = 'FORD'
0130     DISPLAY NAME (0110) MODEL (0120)
0140   END-FIND
0150 END-FIND
...
```

参照するソースコード行番号の変更

Natural ソースプログラムの行番号が変更されると、ステートメントを参照する 4 桁のソースコード行番号（「*表記 (r) を使用したデータベースフィールドの参照*」を参照）も変更されます。 ユーザーの使いやすさ、読みやすさ、およびデバッグのしやすさのために、ステートメント、英数字定数、またはコメントで行われるソースコード行番号への参照はすべて、番号が振り直されます。 ステートメントや英数字定数内のソースコード行番号参照の位置（先頭、中間、最後）は影響しません。

有効なソースコード行番号参照として認識され、番号が振り直されるパターンは以下のとおりです（*nnnn* は 4 桁の数字）。

パターン	ステートメントの例
(*nnnn*)	ESCAPE BOTTOM (0150)
(*nnnn* /	DISPLAY ADDRESS-LINE(0010/1:5)
(*nnnn*,	DISPLAY NAME(0010,A10/1:5)

左側のカッコまたは 4 桁の数字 *nnnn* の後に空白がある、または 4 桁の数字 *nnnn* の後にピリオドがある場合、そのパターンは有効なソースコード行参照とみなされません。

英数字定数内の4桁の数字が不用意に変更されないようにするには、定数を複数に分割し、ハイフンを使用して単一の値に連結する必要があります。

例：

```
Z := 'XXXX (1234,00) YYYY'
```

上記の例は、以下のように変更します。

```
Z := 'XXXX (1234' - ',00) YYYY'
```

ユーザー定義変数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は、変数名の後のカッコ内に指定します。

固定長の変数は、以下のフォーマットおよび対応する長さで定義できます。

ダイナミック変数のフォーマットおよび長さについては、「*ダイナミック変数の定義*」を参照してください。

フォーマット	Explanation	定義可能な長さ	内部的な長さ（バイト）
A	英数字	1〜1073741824（1 GB）	1 - 1073741824
B	バイナリ	1〜1073741824（1 GB）	1 - 1073741824
C	属性制御	-	2
D	日付	-	4
F	浮動小数点	4 または 8	4 または 8
I	整数	1、2 または 4	1、2 または 4
L	論理	-	1
N	数値（アンパック）	1 - 29	1 - 29

フォーマット	Explanation	定義可能な長さ	内部的な長さ（バイト）
P	パック型数値	1 - 29	1 - 15
T	時刻	-	7
U	Unicode（UTF-16）	1～536870912（0.5 GB）	2 - 1073741824

長さは、フォーマットを指定した場合にのみ指定できます。 フォーマットによっては、長さを明示的に指定する必要のないものもあります（上記の表を参照）。

フォーマット N または P で定義するフィールドには、*nn.m* という形式で小数点の位置を指定できます。 *nn* は小数点の前の桁数を、*m* は小数点の後の桁数を表します。 *nn* と *m* を合計した値は 29 を超えることはできません。また、*m* の値は 7 を超えることはできません。

最大の「定義可能な長さ」（英数字、バイナリ、Unicodeフィールドの場合は1GB）は、Natural コンパイラの制限によるものです。 ただし、実際には、データストレージとして取得可能なメモリ量はずっと少なくなります。 特に 「Natural スレッド」 環境で実行する場合、セッションのサイズはユーザーエリアに依存します。したがって、データエリア内のユーザーフィールドのサイズは、マクロ `NTSWPRM` 内のキーワードパラメータ `MAXSIZE` で定義されている値に制限されます。

注意:

1. フォーマット P のユーザー定義変数を `DISPLAY`、`WRITE`、`INPUT` の各ステートメントで出力する場合、Natural によって、出力のために内部的にフォーマット N に変換されます。
2. レポーティングモードでは、ユーザー定義変数のフォーマットおよび長さを指定しません。 デフォルトの割り当てがプロファイル／セッションパラメータ `FS` によって無効化されていない限り、デフォルトのフォーマット／長さの N7 が使用されます。

データベースフィールドの場合、DDM でフィールドに対して定義されているフォーマット／長さが適用されます。 レポーティングモードでは、データベースフィールドに対し、異なるフォーマット／長さをプログラム内で定義することもできます。

ストラクチャードモードでは、データエリア定義または `DEFINE DATA` ステートメントでのみ、フォーマットおよび長さを指定できます。

ストラクチャードモードでのフォーマット／長さの定義例：

```
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
1 #NEW-SALARY (N6.2)
END-DEFINE
...
FIND EMPLOY-VIEW ...
...
```

```
COMPUTE #NEW-SALARY = ...
...
```

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ていなければ、プログラムの本体でフォーマット／長さを定義できます。

レポーティングモードでのフォーマット／長さの定義例：

```
...
...
 FIND EMPLOYEES
... ... COMPUTE #NEW-SALARY(N6.2) = ...
...
```

特殊フォーマット

英数字（A）、数値（B、F、I、N、P）の各標準フォーマットの他に、Natural では以下の特殊フォーマットがサポートされています。

- フォーマット C - 属性制御
- フォーマット D - 日付、およびフォーマット T - 時刻
- フォーマット L - 論理
- フォーマット - ハンドル

フォーマット C - 属性制御

フォーマット C で定義されている変数は、DISPLAY、INPUT、WRITE の各ステートメントで使用されているフィールドに属性をダイナミックに割り当てるために使用できます。

フォーマットCの変数には、長さを指定できません。この変数には常に2バイトの長さがNaturalによって割り当てられます。

例：

```
DEFINE DATA LOCAL
1 #ATTR (C)
1 #A (N5)
END-DEFINE
...
MOVE (AD=I CD=RE) TO #ATTR
INPUT #A (CV=#ATTR)
...
```

詳細については、セッションパラメータ CV の説明を参照してください。

フォーマット D - 日付、およびフォーマット T - 時刻

フォーマット D および T で定義されている変数は、日付および時刻の演算や表示に使用できます。 フォーマット D には、日付情報のみを指定できます。 フォーマット T には、日付および時刻の両方の情報を指定できます。つまり、日付情報は時刻情報のサブセットです。 時刻は、1/10 秒単位でカウントされます。

フォーマット D および T の変数には、長さを指定できません。 フォーマット D の変数には常に 4 バイトの長さ（P6）が、フォーマット T の変数には常に 7 バイトの長さ（P12）が、Natural によって割り当てられます。 プロファイルパラメータ MAXYEAR が 9999 に設定されている場合、フォーマット D の変数には常に 4 バイトの長さ（P7）が、フォーマット T の変数には常に 7 バイトの長さ（P13）が、Natural によって割り当てられます。

例：

```
DEFINE DATA LOCAL
1 #DAT1 (D)
END-DEFINE
*
MOVE *DATX TO #DAT1
ADD 7 TO #DAT1
WRITE '=' #DAT1
END
```

詳細については、セッションパラメータ EM と、システム変数 *DATX および *TIMX の説明を参照してください。

日付フィールドの値は、1582 年 1 月 1 日～2699 年 12 月 31 日の範囲内の値である必要があります。

フォーマット L - 論理

フォーマット L で定義されている変数は、論理条件の基準として使用できます。 この変数の値は、TRUE または FALSE になります。

フォーマット L の変数には、長さを指定できません。 フォーマット L の変数には常に 1 バイトの長さが Natural によって割り当てられます。

例：

```
DEFINE DATA LOCAL
1 #SWITCH(L)
END-DEFINE
MOVE TRUE TO #SWITCH
...
IF #SWITCH
   ...
   MOVE FALSE TO #SWITCH
ELSE
   ...
   MOVE TRUE TO #SWITCH
END-IF
```

論理値表示の詳細については、セッションパラメータ EM の説明を参照してください。

フォーマット - ハンドル

HANDLE OF OBJECT として定義されている変数は、オブジェクトハンドルとして使用できます。

オブジェクトハンドルの詳細については、「*NaturalX*」を参照してください。

インデックス表記

インデックス表記は、配列フィールドに対して使用します。

インデックス表記には、整数値定数またはユーザー定義変数を使用できます。 ユーザー定義変数は、フォーマットN（数値）、P（パック型）、I（整数）、B（バイナリ）のいずれかを使用して指定できます。フォーマット B は、長さが 4 以下の場合にのみ使用できます。

システム変数、システム関数、修飾された変数は、インデックス表記に使用できません。

配列の定義例：

1. #ARRAY (3)
   オカレンス数 3 の 1 次元配列を定義します。
2. FIELD (*label*.,A20/5) または*label*.FIELD(A20/5)
   label. でマークしたステートメントを参照するデータベースフィールドを基にした、英数字フォーマットで長さ 20、オカレンス数 5 の配列を定義します。

3. #ARRAY (N7.2/1:5,10:12,1:4)
   最初のオカレンス数が 5、2 番目のオカレンス数が 3、3 番目のオカレンス数が 4 である、フォーマット／長さが N7.2 の 3 次元配列を定義します。
4. FIELD (*label*./i:i + 5) または*label*.FIELD(i:i + 5)
   label. でマークしたステートメントを参照するデータベースフィールドを基にした配列を定義します。

   FIELDは、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のフィールドを表します。*i* を使用して、データベースオカレンス内のインデックスのオフセットを指定します。 プログラム内では、この配列のサイズはオカレンス数 6（i:i + 5）と定義されます。 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のオカレンスに対する、プログラムの配列の位置を指定するには、データベースインデックスのオフセットを変数として指定します。 *i* を再設定すると、GET または GET SAME ステートメントを使用してデータベースに新規にアクセスする必要があります。

Natural では、インデックスが 1 で始まらない配列を定義できます。 ランタイムに Natural によって、参照で指定しているインデックス値が、定義で指定されている次元の最大サイズを超えていないかどうかがチェックされます。

注意:

1. 以前の Natural バージョンとの互換性を維持するために、コロン（:）の代わりにハイフン（-）を使用して配列の範囲を指定することもできます。
2. ただし、両方の表記を混在させることはできません。
3. ハイフン表記は、レポーティングモードでのみ使用できます。ただし、DEFINE DATA ステートメントでは使用できません。

インデックスの最大値は 1,073,741,824 です。 プログラミングオブジェクトごとのデータエリアの最大サイズは 1,073,741,824 バイト（1 GB）です。

注意: メインフレーム用の Natural バージョン 4.1 との互換性を維持するには：CMPO プロファイルパラメータまたは NTCMPO マクロの V41COMP コンパイルオプションを使用して、メインフレーム用の Natural バージョン 4.1 に適用可能な値になるよう、これらの最大値を小さくします。

添字の参照には、プラス（+） およびマイナス（-）演算子を使用した単純な演算式を使用できます。 演算式を添字として使用する場合、これらの演算子の前後に空白を入れる必要があります。

グループ構造内の配列は、Naturalによって、グループのオカレンスごとではなくフィールドごとに分解されます。

グループ配列の分解例：

```
DEFINE DATA LOCAL
 1 #GROUP (1:2)
   2 #FIELDA (A5/1:2)
   2 #FIELDB (A5)
 END-DEFINE
 ...
```

上記のように定義されているグループを、以下の WRITE ステートメントで出力します。

```
WRITE #GROUP (*)
```

オカレンスは、以下の順序で出力されます。

```
#FIELDA(1,1) #FIELDA(1,2) #FIELDA(2,1) #FIELDA(2,2) #FIELDB(1) #FIELDB(2)
```

以下の順序ではありません。

```
#FIELDA(1,1) #FIELDA(1,2) #FIELDB(1) #FIELDA(2,1) #FIELDA(2,2) #FIELDB(2)
```

配列の参照例：

1. #ARRAY (1)
   1 次元配列の最初のオカレンスを参照します。
2. #ARRAY (7:12)
   1 次元配列の 7～12 番目のオカレンスを参照します。
3. #ARRAY (i + 5)
   1 次元配列の i + 5 番目のオカレンスを参照します。
4. #ARRAY (5,3:7,1:4)
   3 次元配列の、第 1 次元の 5 番目のオカレンス、第 2 次元の 3～7 番目のオカレンス（オカレンス数5）、第3次元のオカレンス1～4番目のオカレンス（オカレンス数4）を参照します。
5. アスタリスクは、次元内のすべてのオカレンスを参照するために使用できます。

   ```
   DEFINE DATA LOCAL
   1 #ARRAY1 (N5/1:4,1:4)
   1 #ARRAY2 (N5/1:4,1:4)
   END-DEFINE
   ...
   ADD #ARRAY1 (2,*) TO #ARRAY2 (4,*)
   ...
   ```

配列オカレンスの前のスラッシュの使用

変数名の後にカッコで囲んだ4桁の数字を指定すると、この番号はステートメントに対する行番号参照だと Natural によって解釈されます。 したがって、4 桁の配列オカレンスを指定する場合、以下の例のように、番号の前にスラッシュ（/）を指定して、その番号が配列オカレンスであることを示す必要があります。

```
#ARRAY(/1000)
```

以下のように指定しないでください。

```
#ARRAY(1000)
```

後者の例は、ソースコード行 1000 への参照と解釈されます。

添字の変数名がフォーマット／長さ指定と間違って解釈される可能性がある場合、スラッシュ（/）を指定して、添字が指定されていることを示す必要があります。 例えば、配列のオカレンスを N7 という変数の値で定義する場合、以下のようにオカレンスを指定する必要があります。

```
#ARRAY (/N7)
```

以下のように指定しないでください。

```
#ARRAY (N7)
```

後者の例は、7 バイトの数値フィールド定義と解釈されます。

データベース配列の参照

以下では次のトピックについて説明します。

- 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フィールドの参照
- 定数を使用して定義されている配列の参照
- 変数を使用して定義されている配列の参照
- 複数定義配列の参照

注意: 以下のサンプルプログラムを実行する前に、SYSEXPG ライブラリ内の INDEXTST プログラムを実行して、10 種類の言語コードを使用するサンプルレコードを作成してください。

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フィールドの参照

ビュー／DDM 内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は、さまざまなインデックス表記を使用して定義および参照できます。

例えば、以下のように、データベースレコード内の同じ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の 1～10 番目の値、および I～I + 10 番目の値を参照できます。

```
DEFINE DATA LOCAL
1 I (I2)
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 LANG (1:10)
  2 LANG (I:I+10)
END-DEFINE
```

または

```
RESET I (I2)
...
READ EMPLOYEES
OBTAIN LANG(1:10) LANG(I:I+10)
```

注意:

1. 同じ下限のインデックス値を使用できるのは、配列ごとに一度だけです。これは、変数インデックスだけでなく定数インデックスの場合も同様です。
2. 変数インデックスを使用した配列定義では、変数自身を使用して下限を、下限と同じ変数 + 定数を使用して上限を指定する必要があります。

定数を使用して定義されている配列の参照

定数を使用して定義されている配列は、定数または変数のいずれかを使用して参照できます。ただし、配列の上限を超えることはできません。 定数を使用すると、コンパイル時に Natural によって、上限を超えているかどうかがチェックされます。

レポーティングモードの例：

```
** Example 'INDEX1R': Array definition with constants  (reporting mode)
************************************************************************
*
READ (1) EMPLOYEES WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  OBTAIN LANG (1:10)
  /*
  WRITE 'LANG(1:10):' LANG (1:10) //
  WRITE 'LANG(1)   :' LANG (1)    /  'LANG(5:9) :' LANG (5:9)
LOOP
*
END
```

ストラクチャードモードの例：

```
** Example 'INDEX1S': Array definition with constants (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 LANG (1:10)
END-DEFINE
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(1:10):' LANG (1:10) //
  WRITE 'LANG(1)   :' LANG (1)    /  'LANG(5:9) :' LANG (5:9)
END-READ
END
```

定数を使用して複数回定義されてい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を、変数を使用して参照する場合、以下のような構文を使用します。

レポーティングモードの例：

```
** Example 'INDEX2R': Array definition with constants (reporting mode)
**                   (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (1:5)
  2 LANG (4:8)
END-DEFINE
*
```

```
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  DISPLAY 'NAME'         NAME
          'LANGUAGE/1:3' LANG (1.1:3)
          'LANGUAGE/6:8' LANG (4.3:5)
LOOP
*
END
```

ストラクチャードモードの例：

```
** Example 'INDEX2S': Array definition with constants (structured mode)
**                    (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (1:5)
  2 LANG (4:8)
END-DEFINE
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  DISPLAY 'NAME'         NAME
          'LANGUAGE/1:3' LANG (1.1:3)
          'LANGUAGE/6:8' LANG (4.3:5)
END-READ
*
END
```

変数を使用して定義されている配列の参照

変数を使用して定義されている配列内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は、同じ変数を使用して参照する必要があります。

レポーティングモードの例：

```
** Example 'INDEX3R': Array definition with variables (reporting mode)
************************************************************************
RESET I (I2)
*
I := 1
READ (1) EMPLOYEES WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  OBTAIN LANG (I:I+10)
  /*
  WRITE 'LANG(I)      :' LANG (I) /
        'LANG(I+5:I+7):' LANG (I+5:I+7)
LOOP
```

```
*
END
```

ストラクチャードモードの例：

```
** Example 'INDEX3S': Array definition with variables (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 I (I2)
*
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (I:I+10)
END-DEFINE
*
I := 1
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(I)      :' LANG (I) /
        'LANG(I+5:I+7):' LANG (I+5:I+7)
END-READ
END
```

別のインデックスを使用すると、変数インデックスを使用した配列の最初の定義を明確に参照できます。 これは、以下のようなインデックス表現を使用すると実行できます。

レポーティングモードの例：

```
** Example 'INDEX4R': Array definition with variables (reporting mode)
************************************************************************
RESET I (I2) J (I2)
*
I := 2
J := 3
*
READ (1) EMPLOYEES WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  OBTAIN LANG (I:I+10)
  /*
  WRITE 'LANG(I.J)  :' LANG (I.J) /
        'LANG(I.1:5):' LANG (I.1:5)
LOOP
*
END
```

ストラクチャードモードの例：

```
** Example 'INDEX4S': Array definition with variables (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 I (I2)
1 J (I2)
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (I:I+10)
END-DEFINE
*
I := 2
J := 3
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(I.J)  :' LANG (I.J) /
        'LANG(I.1:5):' LANG (I.1:5)
END-READ
END
```

式 `I.` は、その配列定義への明示的な参照を作成し、「位置」 を読み込み配列の範囲（`LANG(I.1:5)`）内の最初の値とするために使用します。

データベースにアクセスした時点での `I` の値によって、データベース配列の最初のオカレンスが決まります。

複数定義配列の参照

複数定義配列の場合は通常、適切な配列の範囲を明確に参照できるように、インデックス表現に修飾子を使用する必要があります。

レポーティングモードの例：

```
** Example 'INDEX5R': Array definition with constants (reporting mode)
**                   (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL                   /* For reporting mode programs
1 EMPLOY-VIEW VIEW OF EMPLOYEES     /* DEFINE DATA is recommended
  2 NAME                            /* to use multiple definitions
  2 CITY                            /* of same database field
  2 LANG (1:10)
  2 LANG (5:10)
*
1 I (I2)
1 J (I2)
END-DEFINE
*
```

```
I := 1
J := 2
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(1.1:10) :' LANG (1.1:10) /
        'LANG(1.I:I+2):' LANG (1.I:I+2) //
  WRITE 'LANG(5.1:5)  :' LANG (5.1:5)  /
        'LANG(5.J)    :' LANG (5.J)
LOOP
END
```

ストラクチャードモードの例：

```
** Example 'INDEX5S': Array definition with constants (structured mode)
**                    (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (1:10)
  2 LANG (5:10)
*
1 I (I2)
1 J (I2)
END-DEFINE
*
*
I := 1
J := 2
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(1.1:10) :' LANG (1.1:10) /
        'LANG(1.I:I+2):' LANG (1.I:I+2) //
  WRITE 'LANG(5.1:5)  :' LANG (5.1:5)  /
        'LANG(5.J)    :' LANG (5.J)
END-READ
END
```

インデックス変数を使用して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定義されている場合も、同様の構文を使用します。

レポーティングモードの例：

```
** Example 'INDEX6R': Array definition with variables (reporting mode)
**                    (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 I (I2) INIT <1>
1 J (I2) INIT <2>
1 N (I2) INIT <1>
1 EMPLOY-VIEW VIEW OF EMPLOYEES    /* For reporting mode programs
  2 NAME                           /* DEFINE DATA is recommended
  2 CITY                           /* to use multiple definitions
  2 LANG (I:I+10)                  /* of same database field
  2 LANG (J:J+5)
  2 LANG (4:5)
*
END-DEFINE
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
*
  WRITE 'LANG(I.I)    :' LANG (I.I) /
        'LANG(1.I:I+2):' LANG (I.I:I+10) //
*
  WRITE 'LANG(J.N)    :' LANG (J.N) /
        'LANG(J.2:4)  :' LANG (J.2:4) //
*
  WRITE 'LANG(4.N)    :' LANG (4.N) /
        'LANG(4.N:N+1):' LANG (4.N:N+1) /
LOOP
END
```

ストラクチャードモードの例：

```
** Example 'INDEX6S': Array definition with variables (structured mode)
**                    (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 I (I2) INIT <1>
1 J (I2) INIT <2>
1 N (I2) INIT <1>
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (I:I+10)
  2 LANG (J:J+5)
  2 LANG (4:5)
*
END-DEFINE
*
```

```
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
*
  WRITE 'LANG(I.I)    :' LANG (I.I) /
        'LANG(1.I:I+2):' LANG (I.I:I+10) //
*
  WRITE 'LANG(J.N)    :' LANG (J.N) /
        'LANG(J.2:4)  :' LANG (J.2:4) //
*
  WRITE 'LANG(4.N)    :' LANG (4.N) /
        'LANG(4.N:N+1):' LANG (4.N:N+1) /
END-READ
END
```

データベース配列の内部カウントの参照（C* 表記）

レコードにいくつの値／オカレンスが存在するかが不明な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参照が必要になることがあります。 Adabas は、各マルチプルバリューフィールドの値の個数および各ピリオディックグループのオカレンス数の内部カウントを管理します。 このカウントは、フィールド名の直前に C* を指定することにより参照できます。

Adabas 以外のデータベースに関する注意事項：

SQL	SQL データベースでは、C* 表記は使用できません。
VSAM **DL/I**	VSAM データベースおよび DL/I データベースでは、C* 表記は、値／オカレンスの数ではなく、DDM（MAXOCC）に定義されている値／オカレンスの最大数を返します。

『エディタ』ドキュメントに記載されている、データエリア行コマンド .* の説明も参照してください。

C* フィールドで明示的に宣言できるフォーマットおよび長さは、以下のいずれかです。

- 2 バイト（I2）または 4 バイト（I4）の長さの整数値（I）
- 整数の桁数のみ（精度なし）が指定されている数値（N）またはパック型（P）。例：(N3)。

フォーマットおよび長さが明示的に指定されていない場合、フォーマット／長さ（N3）がデフォルトとして使用されます。

例：

C*LANG	マルチプルバリューフィールド LANG の値の個数を返します。
C*INCOME	ピリオディックグループ INCOME のオカレンス数を返します。
C*BONUS(1)	ピリオディックグループの最初のオカレンスのマルチプルバリューフィールド BONUS 値の個数を返します（BONUS はピリオディックグループ内のマルチプルバリューフィールドとします）。

C* 変数を使用したプログラム例：

```
** Example 'CNOTX01': C* Notation
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 C*INCOME
  2 INCOME
    3 SALARY  (1:5)
    3 C*BONUS (1:2)
    3 BONUS   (1:2,1:2)
  2 C*LANG
  2 LANG      (1:2)
*
1 #I (N1)
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY CITY
  /*
  WRITE NOTITLE 'NAME:' NAME /
        'NUMBER OF LANGUAGES SPOKEN:' C*LANG 5X
        'LANGUAGE 1:' LANG (1) 5X
        'LANGUAGE 2:' LANG (2)
  /*
  WRITE 'SALARY DATA:'
  FOR #I FROM 1 TO C*INCOME
    WRITE 'SALARY' #I SALARY (1.#I)
  END-FOR
  /*
  WRITE 'THIS YEAR BONUS:' C*BONUS(1)  BONUS (1,1) BONUS (1,2)
      / 'LAST YEAR BONUS:' C*BONUS(2)  BONUS (2,1) BONUS (2,2)
  SKIP 1
END-READ
END
```

プログラム CNOTX01 の出力：

```
NAME: SENKO
NUMBER OF LANGUAGES SPOKEN:    1     LANGUAGE 1: ENG     LANGUAGE 2:
SALARY DATA:
SALARY  1      36225
SALARY  2      29900
SALARY  3      28100
SALARY  4      26600
SALARY  5      25200
THIS YEAR BONUS:    0          0          0
LAST YEAR BONUS:    0          0          0

NAME: CANALE
NUMBER OF LANGUAGES SPOKEN:    2     LANGUAGE 1: FRE     LANGUAGE 2: ENG
SALARY DATA:
SALARY  1     202285
THIS YEAR BONUS:    1      23000          0
LAST YEAR BONUS:    0          0          0
```

ピリオディックグループ内のマルチプルバリューフィールドに対する C*

ピリオディックグループ内のマルチプルバリューフィールドに対し、インデックス範囲を使用して C* 変数を定義することもできます。

以下の例は、ピリオディックグループ INCOME の一部分であるマルチプルバリューフィールド BONUS を使用しています。 以下の 3 つの例は、いずれも同じ結果になります。

例 **1 -** レポーティングモード：

```
** Example 'CNOTXO2': C* Notation (multiple-value fields)
************************************************************************
*
LIMIT 2
READ EMPLOYEES BY CITY
  OBTAIN C*BONUS (1:3)
         BONUS   (1:3,1:3)
  /*
  DISPLAY NAME C*BONUS (1:3) BONUS (1:3,1:3)
LOOP
*
END
```

例 **2 -** ストラクチャードモード：

```
** Example 'CNOTX03': C* Notation (multiple-value fields)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 INCOME    (1:3)
    3 C*BONUS
    3 BONUS   (1:3)
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY CITY
  /*
  DISPLAY NAME C*BONUS (1:3) BONUS (1:3,1:3)
END-READ
*
END
```

例 **3 -** ストラクチャードモード：

```
** Example 'CNOTX04': C* Notation (multiple-value fields)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 C*BONUS  (1:3)
  2 INCOME   (1:3)
    3 BONUS  (1:3)
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY CITY
  /*
  DISPLAY NAME C*BONUS (*) BONUS (*,*)
END-READ
*
END
```

注意: Adabas フォーマットバッファではカウントフィールドの範囲指定が認められていないため、個別のフィールドとして生成されます。したがって、大規模な配列に対して C* インデックス範囲を指定すると、Adabas フォーマットバッファがオーバーフローする可能性があります。

データ構造の条件指定

参照するフィールドを指定するために、そのフィールドを修飾できます。つまり、フィールド名の前に、そのフィールドが配置されているレベル 1 のデータ要素名とピリオドを指定します。

フィールド名によってフィールドを一意に識別できない場合（複数のグループ／ビューで同じ名前のフィールド名が使用されている場合など）、参照するフィールドを修飾する必要があります。

レベル 1 のデータ要素とフィールド名の組み合わせは一意である必要があります。

例：

```
DEFINE DATA LOCAL
1 FULL-NAME
  2 LAST-NAME (A20)
  2 FIRST-NAME (A15)
1 OUTPUT-NAME
  2 LAST-NAME (A20)
  2 FIRST-NAME (A15)
END-DEFINE
...
MOVE FULL-NAME.LAST-NAME TO OUTPUT-NAME.LAST-NAME
...
```

修飾語はレベル 1 のデータ要素である必要があります。

例：

```
DEFINE DATA LOCAL
1 GROUP1
  2 SUB-GROUP
    3 FIELD1 (A15)
    3 FIELD2 (A15)
END-DEFINE
...
MOVE 'ABC' TO GROUP1.FIELD1
...
```

データベースフィールドの条件指定

同じ名前のユーザー定義変数とデータベースフィールドを使用している場合（好ましくない状態）、参照するときにデータベースフィールドを修飾する必要があります。

注意: 参照するときにデータベースフィールドを修飾しない場合、代わりにユーザー定義変数が参照されます。

ユーザー定義変数の例

```
DEFINE DATA LOCAL
1 #A1 (A10)       /* Alphanumeric, 10 positions.
1 #A2 (B4)        /* Binary, 4 positions.
1 #A3 (P4)        /* Packed numeric, 4 positions and 1 sign position.
1 #A4 (N7.2)      /* Unpacked numeric,
                  /* 7 positions before and 2 after decimal point.
1 #A5 (N7.)       /* Invalid definition!!!
1 #A6 (P7.2)     /* Packed numeric, 7 positions before and 2 after decimal point
                  /* and 1 sign position.
1 #INT1 (I1)      /* Integer, 1 byte.
1 #INT2 (I2)      /* Integer, 2 bytes.
1 #INT3 (I3)      /* Invalid definition!!!
1 #INT4 (I4)      /* Integer, 4 bytes.
1 #INT5 (I5)      /* Invalid definition!!!
1 #FLT4 (F4)      /* Floating point, 4 bytes.
1 #FLT8 (F8)      /* Floating point, 8 bytes.
1 #FLT2 (F2)      /* Invalid definition!!!
1 #DATE (D)       /* Date (internal format/length P6).
1 #TIME (T)       /* Time (internal format/length P12).
1 #SWITCH (L)     /* Logical, 1 byte (TRUE or FALSE).
                  /*
END-DEFINE
```

17 ダイナミック変数およびフィールドについて

ダイナミック変数の用途

アプリケーションの開発時に大きなデータ構造（画像、音声、ビデオなど）の最大サイズが正確にわからない場合があるため、Naturalでは、英数字変数およびバイナリ変数をDYNAMICという属性を使用して定義できるようにしています。この属性で定義されている変数の値スペースは、実行時に必要となったとき（割り当て操作#picture1 := #picture2を実行するときなど）にダイナミックに拡張されます。つまり、Naturalでは、開発時に制限を定義せずに、大きなバイナリデータ構造および英数字データ構造を処理できます。ダイナミック変数の割り当て時には当然、使用可能なメモリ量の制限を受けます。ダイナミック変数の割り当てに対し、オペレーティングシステムからメモリ不足という結果が返された場合、ON ERRORステートメントを使用してこのエラー条件をインターセプトできます。インターセプトしない場合、Naturalによってエラーメッセージが返されます。

Naturalシステム変数*LENGTHは、指定されたダイナミック変数に現在使用されている値スペースの長さ（コード単位）を取得するために使用できます。AおよびBフォーマットでは、1つのコード単位のサイズは1バイトです。Uフォーマットでは、1つのコード単位のサイズは2バイトです（UTF-16）。*LENGTHは、ダイナミック変数を使用する割り当て中に、Naturalによってソースオペランドの長さに自動的に設定されます。したがって、*LENGTH(*field*)は、Naturalのダイナミックフィールドまたはダイナミック変数に現在使用されている長さ（コード単位）を返します。

ダイナミック変数のスペースが不要になった場合、REDUCEステートメントまたはRESIZEステートメントを使用して、ダイナミック変数に使用されていたスペースをゼロ（または他の任意のサイズ）に減らすことができます。特定のダイナミック変数に対するメモリの使用量がわかっている場合、EXPANDステートメントを使用して、ダイナミック変数に使用するスペースをその特定のサイズに設定できます。

ダイナミック変数を初期化する必要がある場合、MOVE ALL UNTILステートメントを使用して初期化できます。

ダイナミック変数の定義

アプリケーションの開発時に大きな英数字データ構造またはバイナリデータ構造の実際のサイズが正確にわからない場合があるため、フォーマットA、B、Uのダイナミック変数の定義を、これらの構造を管理するために使用できます。ラージ変数のダイナミックアロケーションおよび拡張（再割り当て）は、アプリケーションプログラミングロジックに対して透過的です。ダイナミック変数は長さを指定せずに定義されます。メモリは、実行時にダイナミック変数がターゲットオペランドとして使用されるときに暗黙的に割り当てられるか、EXPANDステートメントまたはRESIZEステートメントの使用によって明示的に割り当てられます。

ダイナミック変数は、以下の構文を使用して、DEFINE DATA ステートメント内でのみ定義できます。

```
level variable-name( A ) DYNAMIC
level variable-name( B ) DYNAMIC
level variable-name( U ) DYNAMIC
```

以下の制限が、ダイナミック変数に適用されます。

- ダイナミック変数は再定義できません。
- ダイナミック変数は REDEFINE 節には使用できません。

ダイナミック変数の現在の値スペース

ダイナミック変数の現在の値スペースの長さ（コード単位）は、システム変数 *LENGTH から取得できます。 *LENGTH は、割り当て時に自動的に、ソースオペランドの（使用されている）長さに設定されます。

注意: パフォーマンスを考慮して、ダイナミック変数の値を保持するために割り当てられているストレージエリアが *LENGTH の値（プログラマが使用できるサイズ）より大きい場合があります。 *LENGTH が示す使用長を超えて割り当てられているストレージは、対応するダイナミック変数にアクセスしていなくても解放される可能性が常にあるため、信頼しないでください。 現在割り当てられているサイズに関する情報を取得する方法はありません。 これは、内部値です。

*LENGTH(*field*) は、Natural のダイナミックフィールドまたはダイナミック変数の使用長（コード単位）を返します。 A および B フォーマットでは、1 つのコード単位のサイズは 1 バイトです。U フォーマットでは、1つのコード単位のサイズは2バイトです（UTF-16）。 *LENGTH は、ダイナミック変数の現在の使用長を取得するためにのみ使用します。

ダイナミック変数のメモリスペースの割り当て／解放

ステートメント EXPAND、REDUCE、および RESIZE は、ダイナミック変数のメモリスペースを明示的に割り当てたり解放したりするために使用します。

構文：

```
EXPAND [SIZE OF] DYNAMIC [VARIABLE] operand1 TO operand2
REDUCE [SIZE OF] DYNAMIC [VARIABLE] operand1 TO operand2
RESIZE [SIZE OF] DYNAMIC [VARIABLE] operand1 TO operand2
```

- *operand1* はダイナミック変数、*operand2* は 0 以上の整数のサイズ値です。

EXPAND

機能

EXPAND ステートメントは、割り当てられているダイナミック変数（*operand1*）の長さを指定された長さ（*operand2*）に増やすために使用します。

指定されたサイズの変更

ダイナミック変数の現在の使用長（Natural システム変数 *LENGTH によって示される。上記参照）は変更されません。

指定された長さ（*operand2*）が、ダイナミック変数に割り当てられている長さに満たない場合、このステートメントは無視されます。

REDUCE

機能

REDUCE ステートメントは、割り当てられているダイナミック変数（*operand1*）の長さを指定された長さ（*operand2*）に減らすために使用します。

指定された長さ（*operand2*）を超える、ダイナミック変数（*operand1*）に割り当てられたストレージは、ステートメントの実行時または実行後に随時解放されます。

指定された長さの変更

ダイナミック変数の現在の使用長（Natural システム変数 *LENGTH によって示される。上記参照）が指定された長さ（*operand2*）を超えている場合、このダイナミック変数の *LENGTH は指定された長さに設定されます。 変数の内容は切り捨てられますが、変更は行われません。

指定された長さが、現在割り当てられているダイナミック変数のストレージを超えている場合、このステートメントは無視されます。

RESIZE

機能

RESIZE ステートメントは、現在割り当てられているダイナミック変数（*operand1*）の長さを指定された長さ（*operand2*）に調整するために使用します。

指定された長さの変更

指定された長さが、ダイナミック変数の使用長（Natural システム変数 *LENGTH によって示される。上記参照）に満たない場合、必要に応じて、ダイナミック変数の使用長が縮小されます。

指定された長さが、現在割り当てられているダイナミック変数の長さを超えている場合、ダイナミック変数に割り当てられる長さが増やされます。 *LENGTH によって示される、ダイナミック変数の現在の使用長は影響を受けず、そのままになります。

指定された長さが、現在割り当てられているダイナミック変数の長さと同じ場合、RESIZE ステートメントを実行しても効力はありません。

18 ダイナミック変数およびラージ変数の使用

全般的な注意事項

通常、次の規則が適用されます。

- ダイナミック英数字フィールドは、英数字フィールドを使用できる場所であればどこにでも使用できます。
- ダイナミックバイナリフィールドは、バイナリフィールドを使用できる場所であればどこにでも使用できます。
- ダイナミック Unicode フィールドは、Unicode フィールドを使用できる場所であればどこにでも使用できます。

例外

ダイナミック変数は、SORT ステートメント内では使用できません。ダイナミック変数を DISPLAY、WRITE、PRINT、REINPUT、INPUT の各ステートメント内で使用するには、セッションパラメータ AL または EM のいずれかを使用して変数の長さを定義する必要があります。

ダイナミック変数に割り当てられているストレージの使用長（Natural システム変数 *LENGTH によって示される。「*ダイナミック変数の現在の値スペース*」を参照）およびサイズは、その変数がターゲットオペランドとして最初にアクセスされるまではゼロです。 割り当て操作またはその他の操作によって、ダイナミック変数はソースオペランドの正確なサイズに初めて割り当てまたは拡張（再割り当て）されます。

以下のステートメントで変更可能なオペランド（ターゲットオペランド）としてダイナミック変数を使用する場合、ダイナミック変数のサイズを拡張できます。

ASSIGN	*operand1*（割り当て内の応答先オペランド）
CALLNAT	「*ダイナミック変数を使用したパラメータ引き渡し*」を参照してください（AD=O、または対応するパラメータデータエリア内に BY VALUE が存在する場合を除く）。
COMPRESS	*operand2*。「*処理*」を参照してください。
EXAMINE	DELETE REPLACE 節内の *operand1*
MOVE	*operand2*（応答先オペランド）。「*機能*」を参照してください。
PERFORM	（AD=O、または対応するパラメータデータエリア内に BY VALUE が存在する場合以外）
READ WORK FILE	*operand1* および *operand2*。「*ラージおよびダイナミック変数の処理*」を参照してください。
SEPARATE	*operand4*.
SELECT (SQL)	INTO 節内のパラメータ
SEND METHOD	*operand3*（AD=O でない場合）

現時点では、ラージ変数の使用には以下の制限があります。

CALL	各パラメータのサイズは 64 KB 未満です（INTERFACE4 オプションを使用した CALL には制限はありません）。

以下のセクションでは、ダイナミック変数の使用について、例を使用してさらに詳しく説明します。

ダイナミック変数を使用した割り当て

通常は、ソースオペランドの現在の使用長（Natural システム変数 *LENGTH によって示される）で割り当てが行われます。 応答先オペランドがダイナミック変数の場合、ソースオペランドを切り捨てずに移動できるよう、現在割り当てられているサイズが拡張される場合があります。

例：

```
#MYDYNTEXT1 := OPERAND
MOVE OPERAND TO #MYDYNTEXT1
/* #MYDYNTEXT1 IS AUTOMATICALLY EXTENDED UNTIL THE SOURCE OPERAND CAN BE COPIED
```

MOVE ALL、およびダイナミックターゲットオペランドを使用した MOVE ALL UNTIL は、以下のように定義されています。

- MOVE ALL では、ターゲットオペランドの使用長（*LENGTH）に到達するまで、ソースオペランドをターゲットオペランドに繰り返し移動します。 *LENGTH は変更できません。 *LENGTH がゼロの場合、このステートメントは無視されます。
- MOVE ALL *operand1* TO *operand2* UNTIL operand3 では、*operand3* に指定した長さに到達するまで、*operand1* を *operand2* に繰り返し移動します。 *operand3* が *LENGTH(*operand2*) より大きい場合、*operand2* は拡張され、*LENGTH(*operand2*) は *operand3* に設定されます。 *operand3* が *LENGTH(*operand2*) より小さい場合、使用長は *operand3* に縮小されます。 *operand3* が *LENGTH(*operand2*) と同じ場合、MOVE ALL と同じ動作をします。

例：

```
#MYDYNTEXT1 := 'ABCDEFGHIJKLMNO'          /* *LENGTH(#MYDYNTEXT1) = 15
MOVE ALL 'AB' TO #MYDYNTEXT1              /* CONTENT OF #MYDYNTEXT1 =
'ABABABABABABABA';
                                          /* *LENGTH IS STILL 15
MOVE ALL 'CD' TO #MYDYNTEXT1 UNTIL 6      /* CONTENT OF #MYDYNTEXT1 = 'CDCDCD';
                                          /* *LENGTH = 6
MOVE ALL 'EF' TO #MYDYNTEXT1 UNTIL 10    /* CONTENT OF #MYDYNTEXT1 = 'EFEFEFEFEF';
                                          /* *LENGTH = 10
```

ターゲットオペランドがダイナミック変数の場合、MOVE JUSTIFIED はコンパイル時に拒否されます。

MOVE SUBSTR および MOVE TO SUBSTR は実行できます。 ダイナミック変数の使用長（*LENGTH）を超えるサブストリングを参照すると、MOVE SUBSTR はランタイムエラーになります。 *LENGTH + 1 を超えるサブストリングの位置を参照すると、ダイナミック変数の内容に未定義のギャップが生じるため、MOVE TO SUBSTR はランタイムエラーになります。 ターゲットオペランドを MOVE TO SUBSTR で拡張する必要がある場合（第 2 オペランドが *LENGTH+1 に設定されている場合など）、第 3 オペランドは必須です。

有効な構文：

```
#OP2 := *LENGTH(#MYDYNTEXT1)
MOVE SUBSTR (#MYDYNTEXT1, #OP2) TO OPERAND                /* MOVE LAST CHARACTER
TO OPERAND
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2, #LEN_OPERAND)   /* CONCATENATE OPERAND
TO #MYDYNTEXT1
```

無効な構文：

```
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE SUBSTR (#MYDYNTEXT1, #OP2, 10) TO OPERAND          /* LEADS TO RUNTIME ERROR;
UNDEFINED SUB-STRING
#OP2 := *LENGTH(#MYDYNTEXT1 + 10)
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2, #EN_OPERAND)   /* LEADS TO RUNTIME ERROR;
UNDEFINED GAP
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2)               /* LEADS TO RUNTIME ERROR;
UNDEFINED LENGTH
```

割り当ての互換性

例：

```
#MYDYNTEXT1   := #MYSTATICVAR1
#MYSTATICVAR1 := #MYDYNTEXT2
```

ソースオペランドがスタティック変数の場合、ダイナミックな応答先オペランドの使用長（*LENGTH(#MYDYNTEXT1)）はスタティック変数のフォーマット長に設定され、末尾の空白（フォーマット A および U）またはバイナリゼロ（フォーマット B）を含め、この長さでソースオペランドがコピーされます。

応答先オペランドがスタティックでソースオペランドがダイナミックの場合、ダイナミック変数は現在の使用長でコピーされます。 この長さがスタティック変数のフォーマット長より短い場

合、残りの部分は空白（英数字フィールドおよびUnicodeフィールドの場合）またはバイナリゼロ（バイナリフィールドの場合）で埋められます。そうでない場合、値は切り捨てられます。ダイナミック変数の現在の使用長が0の場合、スタティックのターゲットオペランドは空白（英数字フィールドおよびUnicodeフィールドの場合）またはバイナリゼロ（バイナリフィールドの場合）で埋められます。

ダイナミック変数の初期化

ダイナミック変数は、RESETステートメントを使用して、最大で現在の使用長（=*LENGTH）まで、空白（英数字フィールドおよびUnicodeフィールドの場合）またはゼロ（バイナリフィールドの場合）で初期化できます。 *LENGTHは変更できません。

例：

```
DEFINE DATA LOCAL
1 #MYDYNTEXT1   (A)  DYNAMIC
END-DEFINE
#MYDYNTEXT1 := 'SHORT TEXT'
WRITE *LENGTH(#MYDYNTEXT1)          /* USED LENGTH = 10
RESET #MYDYNTEXT1                   /* USED LENGTH = 10, VALUE = 10 BLANKS
```

ダイナミック変数を特定のサイズと値で初期化するには、MOVE ALL UNTILステートメントを使用します。

例：

```
MOVE ALL 'Y' TO #MYDYNTEXT1 UNTIL 15         /* #MYDYNTEXT1 CONTAINS 15 'Y'S, USED
LENGTH = 15
```

ダイナミック英数字変数での文字列操作

変更可能なオペランドがダイナミック変数の場合、切り捨てまたはエラーメッセージを発生させずに操作できるよう、現在割り当てられているサイズが拡張される場合があります。 これは、ダイナミック変数の連結（COMPRESS）および分割（SEPARATE）で有効です。

例：

```
** Example 'DYNAMX01': Dynamic variables (with COMPRESS and SEPARATE)
************************************************************************
DEFINE DATA LOCAL
1 #MYDYNTEXT1 (A)   DYNAMIC
1 #TEXT       (A20)
1 #DYN1       (A)   DYNAMIC
1 #DYN2       (A)   DYNAMIC
1 #DYN3       (A)   DYNAMIC
END-DEFINE
*
MOVE ' HELLO WORLD ' TO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with leading and trailing blanks
*
MOVE ' HELLO WORLD ' TO #TEXT
*
MOVE #TEXT TO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with whole variable length of #TEXT
*
COMPRESS #TEXT INTO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with leading blanks of #TEXT
*
*
#MYDYNTEXT1 := 'HERE COMES THE SUN'
SEPARATE #MYDYNTEXT1 INTO #DYN1 #DYN2 #DYN3 IGNORE
*
WRITE / #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
WRITE #DYN1 (AL=25) 'with length' *LENGTH (#DYN1)
WRITE #DYN2 (AL=25) 'with length' *LENGTH (#DYN2)
WRITE #DYN3 (AL=25) 'with length' *LENGTH (#DYN3)
/*  #DYN1, #DYN2, #DYN3 are automatically extended or reduced
*
EXAMINE #MYDYNTEXT1 FOR 'SUN' REPLACE 'MOON'
WRITE / #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  #MYDYNTEXT1 is automatically extended or reduced
*
END
```

注意: 非ダイナミック変数の場合、エラーメッセージが返されます。

ダイナミック変数を使用した論理条件の基準（LCC）

ダイナミック変数を使用して読み取り専用の操作（比較など）を行う場合、通常は現在の使用長を使用して実行されます。 読み取り操作（変更なし）で使用される場合、ダイナミック変数はスタティック変数と同様に処理されます。

例：

```
IF  #MYDYNTEXT1  =  #MYDYNTEXT2  OR  #MYDYNTEXT1 =  "**"  THEN ...
IF  #MYDYNTEXT1  <  #MYDYNTEXT2  OR  #MYDYNTEXT1 <  "**"  THEN ...
IF  #MYDYNTEXT1  >  #MYDYNTEXT2  OR  #MYDYNTEXT1 >  "**"  THEN ...
```

英数字変数および Unicode 変数の末尾の空白またはバイナリ変数の先頭のバイナリゼロは、スタティック変数とダイナミック変数で同様に処理されます。 例えば、AA および空白が続く AA を値として持つ英数字変数は同一とみなされ、H'00003031' および H'3031' という値を持つバイナリ変数は同一とみなされます。値が完全に同じ場合にのみ比較結果をTRUEとする場合は、ダイナミック変数の使用長も比較する必要があります。 一方の変数ともう一方の変数の値が完全に同じであれば、その使用長もまた同じです。

例：

```
#MYDYNTEXT1  :=  'HELLO'                                    /* USED LENGTH IS 5
#MYDYNTEXT2  :=  'HELLO     '                               /* USED LENGTH IS 10
IF   #MYDYNTEXT1  =  #MYDYNTEXT2   THEN  ...                /* TRUE
IF   #MYDYNTEXT1  =  #MYDYNTEXT2 AND
     *LENGTH(#MYDYNTEXT1) = *LENGTH(#MYDYNTEXT2)  THEN  ...  /* FALSE
```

2つのダイナミック変数は、どちらか短い方の使用長に達するまで1ポジションずつ比較されます（英数字変数の場合は左から右、バイナリ変数の場合は右から左）。 最初に値が異なったポジションで、1番目の変数と2番目の変数のどちらがより大きいか、小さいか、または同じかが判断されます。 変数の短い方の使用長まで値が同じで、長い方の残りの値がダイナミック英数字変数の場合は空白のみ、ダイナミックバイナリ変数の場合はバイナリゼロのみの場合、それらの変数は同じとみなされます。 2つのダイナミック Unicode 変数を比較する場合、両方の値の末尾の空白を削除してから、ICU照合アルゴリズムを使用して2つの値が比較されます。『*Unicodeとコードページのサポート*』ドキュメントの「*論理条件の基準*」も参照してください。

例：

```
#MYDYNTEXT1  :=  'HELLO1'                    /* USED LENGTH IS 6
#MYDYNTEXT2  :=  'HELLO2'                    /* USED LENGTH IS 10
IF #MYDYNTEXT1  <  #MYDYNTEXT2 THEN  ...     /* TRUE
#MYDYNTEXT2  :=  'HALLO'
IF #MYDYNTEXT1  >  #MYDYNTEXT2 THEN  ...     /* TRUE
```

比較の互換性

ダイナミック変数とスタティック変数の比較は、ダイナミック変数間の比較と同じです。 スタティック変数のフォーマット長が使用長として使用されます。

例：

```
#MYSTATTEXT1 :=  'HELLO'                      /* FORMAT LENGTH OF MYSTATTEXT1 IS
A20
#MYDYNTEXT1  :=  'HELLO'                      /* USED LENGTH IS 5
IF  #MYSTATTEXT1  =  #MYDYNTEXT1  THEN  ...   /* TRUE
IF  #MYSTATTEXT1  >  #MYDYNTEXT1  THEN  ...   /* FALSE
```

ダイナミックコントロールフィールドの AT／IF-BREAK

ブレイクコントロールフィールドの元の値との比較は、左から右に向かって1ポジションずつ実行されます。 ダイナミック変数の元の値と新しい値で長さが異なる場合、比較するために長さが短い方の値の右側に空白（ダイナミック英数字変数またはダイナミックUnicode変数の場合）またはバイナリゼロ（バイナリ値の場合）が追加されます。

英数字または Unicode のブレイクコントロールフィールドの場合、比較において末尾の空白は意味を持ちません。つまり、末尾の空白は値の変更を意味しないため、ブレイクは発生しません。

バイナリのブレイクコントロールフィールドの場合、比較において末尾のバイナリゼロは意味を持ちません。つまり、末尾のバイナリゼロは値の変更を意味しないため、ブレイクは発生しません。

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。 また、値渡しの結果では、逆方向にデータが移動します。

参照渡しを使用する場合、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。 値渡し（結果）の場合、すべての組み合わせを使用できます。 以下の表に、有効な組み合わせを示します。

参照渡し

呼び出し元	パラメータ	
	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する必要があります。

値渡し（結果）

呼び出し元	パラメータ	
	スタティック	ダイナミック
スタティック	◯	◯
ダイナミック	◯	◯

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

例 **1**：

```
** Example 'DYNAMX02': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA LOCAL
1 #MYTEXT (A) DYNAMIC
END-DEFINE
*
#MYTEXT := '123456'      /* extended to 6 bytes, *LENGTH(#MYTEXT) = 6
*
CALLNAT 'DYNAMX03' USING #MYTEXT
*
```

```
WRITE *LENGTH(#MYTEXT)          /* *LENGTH(#MYTEXT) = 8
*
END
```

サブプログラム DYNAMX03：

```
** Example 'DYNAMX03': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA PARAMETER
1 #MYPARM (A) DYNAMIC  BY VALUE RESULT
END-DEFINE
*
WRITE *LENGTH(#MYPARM)                  /* *LENGTH(#MYPARM) = 6
#MYPARM := '1234567'                    /* *LENGTH(#MYPARM) = 7
#MYPARM := '12345678'                   /* *LENGTH(#MYPARM) = 8
EXPAND DYNAMIC VARIABLE #MYPARM TO 10   /* 10 bytes are allocated
*
WRITE *LENGTH(#MYPARM)                  /* *LENGTH(#MYPARM) = 8
*
/* content of #MYPARM is moved back to #MYTEXT
/* used length of #MYTEXT = 8
*
END
```

例 **2**：

```
** Example 'DYNAMX04': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA LOCAL
1 #MYTEXT (A) DYNAMIC
END-DEFINE
*
#MYTEXT := '123456'      /* extended to 6 bytes, *LENGTH(#MYTEXT) = 6
*
CALLNAT 'DYNAMX05' USING #MYTEXT
*
WRITE *LENGTH(#MYTEXT)          /* *LENGTH(#MYTEXT) = 8
                                /* at least 10 bytes are
                                /* allocated (extended in DYNAMX05)
*
END
```

サブプログラム DYNAMX05：

```
** Example 'DYNAMX05': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA PARAMETER
1 #MYPARM (A) DYNAMIC
END-DEFINE
*
WRITE *LENGTH(#MYPARM)                  /* *LENGTH(#MYPARM) = 6
#MYPARM := '1234567'                    /* *LENGTH(#MYPARM) = 7
#MYPARM := '12345678'                   /* *LENGTH(#MYPARM) = 8
EXPAND DYNAMIC VARIABLE #MYPARM TO 10   /* 10 bytes are allocated
*
WRITE *LENGTH(#MYPARM)                  /* *LENGTH(#MYPARM) = 8
*
END
```

3GL プログラムに対する CALL

CALL ステートメントで INTERFACE4 オプションを使用すると、ダイナミック変数およびラージ変数を有効に使用できます。このオプションを使用すると、異なるパラメータ構造を使用した3GLプログラムとのインターフェイスになります。

ダイナミックパラメータを使用して 3GL プログラムを呼び出す前に、必要なバッファサイズを必ず割り当てるようにしてください。 これは、EXPAND ステートメントを使用すると明示的に実行できます。

バッファを初期化する必要がある場合、MOVE ALL UNTIL ステートメントを使用することにより、ダイナミック変数を初期値および必要なサイズに設定できます。 Natural には、3GL プログラムでダイナミックパラメータに関する情報を取得し、パラメータデータを返すときに長さを変更できるようにする、一連のファンクションが用意されています。

例：

```
MOVE ALL ' ' TO  #MYDYNTEXT1 UNTIL 10000
  /* a buffer of length 10000 is allocated
  /* #MYDYNTEXT1 is initialized with blanks
  /* and *LENGTH(#MYDYNTEXT1) = 10000
CALL INTERFACE4  'MYPROG'  USING  #MYDYNTEXT1
WRITE *LENGTH(#MYDYNTEXT1)
  /* *LENGTH(#MYDYNTEXT1) may have changed in the 3GL program
```

詳細については、『ステートメント』ドキュメントに記載されている、CALL ステートメントの説明を参照してください。

ラージ変数およびダイナミック変数によるワークファイルへのアクセス

253 以下の長さの固定長の変数と 253 を超える長さのラージ変数の扱いは同じです。

ダイナミック変数は、WRITE WORK FILE ステートメントの実行時に、その実際の長さ（つまり、この変数に対するシステム変数 *LENGTH の値）でデータが書き込まれます。 同じ WRITE WORK FILE ステートメントであっても実行するたびに長さが異なるため、キーワード VARIABLE を指定する必要があります。

タイプ FORMATTED のワークファイルを読み込む場合、ダイナミック変数には、READ WORK FILE ステートメントの実行時に、その実際の長さ（つまり、この変数に対するシステム変数 *LENGTH の値）でデータが読み込まれます。 ダイナミック変数が現在のレコードの残りのデータより長い場合、空白（英数字フィールドおよび Unicode フィールドの場合）またはバイナリゼロ（バイナリフィールドの場合）が埋め込まれます。

タイプ UNFORMATTED のワークファイルを読み込む場合、ダイナミック変数にはワークファイルの残りが読み込まれます。 サイズは適切に調整され、この変数に対するシステム変数 *LENGTH の値に反映されます。

ダイナミック変数の使用によるパフォーマンスへの影響

ダイナミック変数を少量ずつ複数回にわたって拡張する場合（バイト単位など）、必要なストレージの（おおよその）上限がわかっているときは、拡張を繰り返すのではなく EXPAND ステートメントを使用します。 これにより、必要なストレージを調整するための余分なオーバーヘッドを回避できます。

ダイナミック変数が不要になった場合、特に *LENGTH の値が大きい変数の場合は、REDUCE ステートメントまたは RESIZE ステートメントを使用します。 これにより、Natural でストレージを解放または再利用できます。 したがって、全体的なパフォーマンスが向上します。

ダイナミック変数に割り当てられているメモリの量は、REDUCE DYNAMIC VARIABLE ステートメントを使用すると減らすことができます。変数に特定の長さを（再）割り当てするには、EXPAND ステートメントを使用できます （変数を初期化する場合は、MOVE ALL UNTIL ステートメントを使用します）。

例：

```
** Example 'DYNAMXO6': Dynamic variables (allocated memory)
************************************************************************
DEFINE DATA LOCAL
1 #MYDYNTEXT1  (A)  DYNAMIC
1 #LEN         (I4)
END-DEFINE
*
#MYDYNTEXT1 := 'a'     /* used length is 1, value is 'a'
                       /* allocated size is still 1
WRITE *LENGTH(#MYDYNTEXT1)
*
EXPAND DYNAMIC VARIABLE  #MYDYNTEXT1 TO 100
                       /* used length is still 1, value is 'a'
                       /* allocated size is 100
*
CALLNAT 'DYNAMXO5' USING #MYDYNTEXT1
WRITE *LENGTH(#MYDYNTEXT1)
                       /* used length and allocated size
                       /* may have changed in the subprogram
*
#LEN := *LENGTH(#MYDYNTEXT1)
REDUCE DYNAMIC VARIABLE  #MYDYNTEXT1  TO  #LEN
                       /* if allocated size is greater than used length,
                       /* the unused memory is released
*
REDUCE DYNAMIC VARIABLE  #MYDYNTEXT1  TO  0
WRITE *LENGTH(#MYDYNTEXT1)
                       /* free allocated memory for dynamic variable
END
```

ルール：

- ダイナミックオペランドは、適切な場所で使用します。
- メモリ使用量の上限がわかっている場合は、EXPAND ステートメントを使用します。
- ダイナミックオペランドが不要になった場合は、REDUCE ステートメントを使用します。

ダイナミック変数の出力

ダイナミック変数は、以下のような出力ステートメント内で使用できます。

ステートメント	注意点
DISPLAY	これらのステートメントでは、セッションパラメータの AL（出力に対する英数字長）またはEM（編集マスク）を使用して、ダイナミック変数の入出力フォーマットを設定する必要があります。
WRITE	
INPUT	
REINPUT	--
PRINT	PRINT ステートメントの出力はフォーマットが指定されていないため、AL および EM パラメータを使用して、PRINT ステートメント内のダイナミック変数の出力フォーマットを設定する必要はありません。 したがって、これらのパラメータは無視されます。

ダイナミック X-array

ダイナミックX-arrayの割り当てでは、最初にオカレンス数を指定し、後でそのオカレンス数を拡張できます。

例：

```
DEFINE DATA LOCAL
1 #X-ARRAY(A/1:*)  DYNAMIC
END-DEFINE
*
EXPAND  ARRAY   #X-ARRAY TO (1:10)  /* Current boundaries (1:10)
#X-ARRAY(*) := 'ABC'
EXPAND  ARRAY   #X-ARRAY TO (1:20)  /* Current boundaries (1:20)
#X-ARRAY(11:20) := 'DEF'
```

19 ユーザー定義定数

定数は、Naturalプログラム全体で使用できます。このドキュメントでは、サポートされている定数のタイプとその使用方法について説明します。

数値定数

以下では次のトピックについて説明します。

- 数値定数
- 数値定数の検証

数値定数

数値定数には、1～29桁の数字と小数点表記のための特殊文字（ピリオドまたはコンマ）を指定できます。

例：

```
1234   +1234   -1234

12.34  +12.34  -12.34
```

```
MOVE 3 TO #XYZ
COMPUTE #PRICE = 23.34
COMPUTE #XYZ = -103
COMPUTE #A = #B * 6074
```

数値定数は、内部的にはパック型フォーマット（フォーマットP）で表されます。ただし、数値定数が算術演算で使用され、もう1つのオペランドが整数の変数（フォーマットI）の場合は例外的に、数値定数は整数フォーマット（フォーマットI）で表されます。

数値定数の検証

INITオプションを使用したCOMPUTE、MOVE、DEFINE DATAの各ステートメント内で数値定数を使用すると、コンパイル時にNaturalによって、対応するフィールドに対して定数値が適しているかどうかが確認されます。これにより、コンパイル時にエラー条件が検出されるため、ランタイムエラーを回避できます。

英数字定数

以下では次のトピックについて説明します。

英数字定数

英数字定数には、1～1073741824 バイト（1 GB）の英数字を指定できます。

英数字定数は、次のいずれかで囲む必要があります。：アポストロフィ（'）

```
'text'
```

または引用符（"）

```
"text"
```

例：

```
MOVE 'ABC' TO #FIELDX
MOVE '% INCREASE' TO #TITLE
DISPLAY "LAST-NAME" NAME
```

注意: ユーザー定義変数に割り当てるために使用する 1 つの英数字定数は、複数のステートメント行に分割できません。

英数字定数内のアポストロフィ

アポストロフィで囲まれた英数字定数内で 1 つのアポストロフィを表すには、2 つのアポストロフィまたは 1 つの引用符を使用する必要があります。

引用符で囲まれた英数字定数内で 1 つのアポストロフィを表すには、1 つのアポストロフィを指定します。

例：

以下を出力するとします。

```
HE SAID, 'HELLO'
```

以下のいずれの表記も使用できます。

```
WRITE 'HE SAID, ''HELLO'''
WRITE 'HE SAID, "HELLO"'
WRITE "HE SAID, ""HELLO"""
WRITE "HE SAID, 'HELLO'"
```

注意: 上述したように引用符がアポストロフィに変換されない場合、これはプロファイルパラメータ `TQ`（Translate Quotation Marks）の設定が原因です。詳細については、Natural の管理者に問い合わせてください。

英数字定数の連結

ハイフンを使用すると、複数の英数字定数を 1 つの値として連結できます。

例：

```
MOVE 'XXXXXX' - 'YYYYYY' TO #FIELD
MOVE "ABC" - 'DEF' TO #FIELD
```

上記の方法で、英数字定数を **16** 進定数と連結することもできます。

Unicode 定数

以下では次のトピックについて説明します。

- Unicode テキスト定数
- Unicode テキスト定数内のアポストロフィ
- Unicode 16 進定数

- Unicode 定数の連結

Unicode テキスト定数

Unicodeテキスト定数は、先頭に文字Uを指定し、その後の文字列を次のいずれかで囲む必要があります。：アポストロフィ（'）

```
U'text'
```

または引用符（"）

```
U"text"
```

例：

```
U'HELLO'
```

コンパイラは、このテキスト定数を Unicode フォーマット（UTF-16）で生成プログラムに格納します。

Unicode テキスト定数内のアポストロフィ

アポストロフィで囲まれたUnicodeテキスト定数内で1つのアポストロフィを表すには、2つのアポストロフィまたは1つの引用符を使用する必要があります。

引用符で囲まれたUnicodeテキスト定数内で1つのアポストロフィを表すには、1つのアポストロフィを指定します。

例：

以下を出力するとします。

```
HE SAID, 'HELLO'
```

以下のいずれの表記も使用できます。

```
WRITE U'HE SAID, ''HELLO'''
WRITE U'HE SAID, "HELLO"'
WRITE U"HE SAID, ""HELLO"""
WRITE U"HE SAID, 'HELLO'"
```

注意: 上述したように引用符がアポストロフィに変換されない場合、これはプロファイルパラメータTQ（Translate Quotation Marks）の設定が原因です。詳細については、Naturalの管理者に問い合わせてください。

Unicode 16 進定数

以下の構文は、Unicode 文字または Unicode 文字列を 16 進表記で指定するために使用します。

```
UH'hhhh...'
```

h は、16 進数の桁（0〜9、A〜F）を表します。 UTF-16 Unicode 文字はダブルバイトであるため、指定する 16 進の文字数は 4 の倍数である必要があります。

例：

以下の例は、文字列 45 を定義しています。

```
UH'00340035'
```

Unicode 定数の連結

Unicode テキスト定数（U）および Unicode 16 進定数（UH）は、連結できます。

有効な例：

```
MOVE U'XXXXXX' - UH'00340035' TO #FIELD
```

Unicode テキスト定数または Unicode 16 進定数は、コードページ英数字定数または H 定数とは連結できません。

無効な例：

```
MOVE U'ABC' - 'DEF' TO #FIELD
MOVE UH'00340035' - H'414243' TO #FIELD
```

その他の有効な例：

```
DEFINE DATA LOCAL
1 #U10 (U10)              /* Unicode variable with 10 (UTF-16) characters, total
byte length = 20
1 #UD  (U)  DYNAMIC        /* Unicode variable with dynamic length
END-DEFINE
*
#U10 := U'ABC'            /* Constant is created as X'004100420043' in the object,
the UTF-16 representation for string 'ABC'.

#U10 := UH'004100420043'   /* Constant supplied in hexadecimal format only,
corresponds to U'ABC'

#U10 := U'A'-UH'0042'-U'C' /* Constant supplied in mixed formats, corresponds to
```

```
U'ABC'.
END
```

日付／時刻の定数

以下では次のトピックについて説明します。

- 日付定数
- 時刻定数
- 拡張時刻定数

日付定数

日付定数は、フォーマット D の変数とともに使用します。

日付定数は、以下の形式で定義します。

D'*yyyy-mm-dd*'	国際式
D'*dd.mm.yyyy*'	ドイツ式
D'*dd/mm/yyyy*'	ヨーロッパ式
D'*mm/dd/yyyy*'	US 日付フォーマット

上記の *dd* は日、*mm* は月、*yyyy* は年を表します。

例：

```
DEFINE DATA LOCAL
1 #DATE (D)
END-DEFINE
...
MOVE D'2004-03-08' TO #DATE
...
```

デフォルトの日付形式は、プロファイルパラメータ DTFORM（日付形式）を使用して、Natural 管理者が制御します。

時刻定数

時刻定数は、フォーマット T の変数とともに使用します。

時刻定数は、以下の形式で定義します。

```
T'hh:ii:ss'
```

上記の *hh* は時間、*ii* は分、*ss* は秒を表します。

例：

```
DEFINE DATA LOCAL
1 #TIME (T)
END-DEFINE
...
MOVE T'11:33:00' TO #TIME
...
```

拡張時刻定数

日付情報は時刻情報のサブセットであるため、時刻変数（フォーマットT）には、日付および時刻の情報を格納できます。ただし、「標準の」時刻定数（接頭辞T）では、時刻変数の時刻情報のみを処理できます。

```
T'hh:ii:ss'
```

拡張時刻定数（接頭辞E）を使用すると、日付情報を含む、時刻変数のすべての内容を処理できます。

```
E'yyyy-mm-dd hh:ii:ss'
```

この点を除き、拡張時刻定数と時刻変数の使用方法は、標準の時刻定数の場合と同じです。

注意: 拡張時刻定数に指定する必要のある日付情報の形式は、プロファイルパラメータ DTFORM の設定によって決まります。 上記の拡張時刻定数は、DTFORM=I（国際式）を想定しています。

16 進の定数

以下では次のトピックについて説明します。

16 進の定数

16 進定数では、標準のキーボード文字で入力できない値を指定できます。

16 進定数には、1～1073741824 バイト（1 GB）の英数字を指定できます。

接頭文字 `H` で、16 進定数を示します。 定数自身は、16 進数を表す文字（0～9、A～F）で構成される文字列をアポストロフィで囲む必要があります。 1 バイトのデータを表すには、2 つの 16 進文字が必要です。

16 進文字表現は、コンピュータが ASCII 文字セットを使用しているか、EBCDIC 文字セットを使用しているかによって異なります。 したがって、16 進定数を別のコンピュータに転送する場合、文字変換が必要になることがあります。

ASCII の例：

```
H'313233'    (equivalent to the alphanumeric constant '123')
H'414243'    (equivalent to the alphanumeric constant 'ABC')
```

EBCDIC の例：

```
H'F1F2F3'    (equivalent to the alphanumeric constant '123')
H'C1C2C3'    (equivalent to the alphanumeric constant 'ABC')
```

16 進定数を別のフィールドに転送する場合は、英数字値（フォーマット A）として扱われます。

フォーマット A、U、B 以外で定義されているフィールドに、英数字値（フォーマット A）をデータ転送することはできません。 したがって、対応する変数がフォーマット A、U、B ではない場合に `DEFINE DATA` ステートメントで 16 進定数を初期値として使用すると、構文エラー NAT0094 で拒否されます。

例：

```
DEFINE DATA LOCAL
1 #I(I2) INIT <H'000F'>      /* causes a NAT0094 syntax error
END-DEFINE
```

16 進定数の連結

定数の間にハイフンを使用すると、16 進定数を連結できます。

ASCII の例：

```
H'414243' - H'444546' (equivalent to 'ABCDEF')
```

EBCDIC の例：

```
H'C1C2C3' - H'C4C5C6' (equivalent to 'ABCDEF')
```

上記の方法で、16 進定数を英数字定数と連結することもできます。

論理定数

論理定数 TRUE および FALSE は、フォーマット L で定義されたフィールドに論理値を割り当てるために使用できます。

例：

```
DEFINE DATA LOCAL
1 #FLAG (L)
END-DEFINE
...
MOVE TRUE TO #FLAG
...
IF #FLAG ...
  statement ...
  MOVE FALSE TO #FLAG
END-IF
 ...
```

浮動小数点定数

浮動小数点定数は、フォーマット F で定義された変数で使用できます。

例：

```
DEFINE DATA LOCAL
1 #FLT1 (F4)
END-DEFINE
...
COMPUTE #FLT1 = -5.34E+2
...
```

属性定数

属性定数は、フォーマットCで定義された変数（制御変数）で使用できます。属性定数は、カッコで囲む必要があります。

次の属性を使用できます。

属性	説明
AD=D	デフォルト
AD=B	点滅
AD=I	高輝度
AD=N	非表示
AD=V	反転
AD=U	下線付き
AD=C	イタリック
AD=Y	ダイナミック属性
AD=P	保護
CD=BL	青
CD=GR	緑
CD=NE	デフォルト色
CD=PI	ピンク
CD=RE	赤
CD=TU	空色
CD=YE	黄色

セッションパラメータ AD および CD の説明も参照してください。

例：

```
DEFINE DATA LOCAL
1 #ATTR (C)
1 #FIELD (A10)
END-DEFINE
...
MOVE (AD=I CD=BL) TO #ATTR
...
INPUT #FIELD (CV=#ATTR)
...
```

ハンドル定数

ハンドル定数 NULL-HANDLE は、オブジェクトハンドルで使用できます。

オブジェクトハンドルの詳細については、「*NaturalX*」を参照してください。

名前付き定数の定義

同じ定数値を何度も使用する必要がある場合、以下のように名前付き定数を定義してメンテナンスの労力を減らすことができます。

- DEFINE DATA ステートメント内でフィールドを定義します。
- フィールドに定数値を割り当てます。
- プログラムで定数値ではなくフィールド名を使用します。

これにより、値を変更する必要があるときには、プログラム内の関連するすべての場所を変更するのではなく、DEFINE DATA ステートメントを1回変更するだけですべての値を変更できます。

DEFINE DATA ステートメントのフィールド定義の後に、キーワード CONSTANT を付けた山カッコ（< と >）内に定数値を指定します。

- 値が英数字の場合、アポストロフィで囲む必要があります。
- 値が Unicode テキストフォーマットの場合、先頭に文字 U を指定し、その後の文字列をアポストロフィで囲む必要があります。
- 値が Unicode 16 進フォーマットの場合、先頭に文字 UH を指定し、その後の文字列をアポストロフィで囲む必要があります。

例：

```
DEFINE DATA LOCAL
1 #FIELDA (N3) CONSTANT <100>
1 #FIELDB (A5) CONSTANT <'ABCDE'>
1 #FIELDC (U5) CONSTANT <U'ABCDE'>
1 #FIELDD (U5) CONSTANT <UH'00410042004300440045'>
END-DEFINE
...
```

プログラム実行中は、名前付き定数の値を変更できません。

20 初期値（および RESET ステートメント）

この章では、次のトピックについて説明します。

ユーザー定義変数／配列のデフォルトの初期値

フィールドに初期値を指定しない場合、そのフォーマットに応じたデフォルトの初期値でフィールドは初期化されます。

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

ユーザー定義変数／配列への初期値の割り当て

DEFINE DATA ステートメントでは、ユーザー定義変数に初期値を割り当てることができます。初期値が英数字の場合、アポストロフィで囲む必要があります。

- 変更可能な初期値の割り当て
- 変更不可能な初期値の割り当て
- 初期値としての Natural システム変数の割り当て
- 英数字変数の初期値としての文字列の割り当て

変更可能な初期値の割り当て

変更可能な初期値を変数／配列に割り当てる場合、DEFINE DATA ステートメントの変数定義の後に、キーワード INIT とともに山カッコ（< と >）で囲んだ初期値を指定します。 割り当てられた値は、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変更できます。

例：

```
DEFINE DATA LOCAL
1 #FIELDA (N3) INIT <100>
1 #FIELDB (A20) INIT <'ABC'>
END-DEFINE
...
```

変更不可能な初期値の割り当て

変数／配列を名前付き定数として扱う場合、DEFINE DATA ステートメントの変数定義の後に、キーワード CONSTANT とともに山カッコ（< と >）で囲んだ初期値を指定します。 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。

例：

```
DEFINE DATA LOCAL
1 #FIELDA (N3) CONST <100>
1 #FIELDB (A20) CONST <'ABC'>
END-DEFINE
...
```

初期値としての Natural システム変数の割り当て

フィールドの初期値として、**Natural** システム変数の値を指定することもできます。

例：

以下の例では、システム変数 *DATX が初期値を指定するために使用されています。

```
DEFINE DATA LOCAL
1 #MYDATE (D) INIT <*DATX>
END-DEFINE
...
```

英数字変数の初期値としての文字列の割り当て

初期値として、特定の1文字または文字列で、変数全体または変数の一部を埋めることができます（英数字変数にのみ有効）。

- フィールド全体を埋める場合
  FULL LENGTH <*character(s)*> オプションを使用して、フィールド全体を特定の文字（列）で埋めます。

  以下の例では、フィールド全体をアスタリスク（*）で埋めています。

```
DEFINE DATA LOCAL
1 #FIELD (A25) INIT FULL LENGTH <'*'>
END-DEFINE
...
```

- フィールドの最初の*n* 文字を埋める場合
  LENGTH *n* <*character(s)*> オプションを使用して、フィールドの先頭 *n* 文字を特定の文字（列）で埋めます。

  以下の例では、フィールドの先頭 4 文字を感嘆符（!）で埋めています。

```
DEFINE DATA LOCAL
1 #FIELD (A25) INIT LENGTH 4 <'!'>
END-DEFINE
...
```

ユーザー定義変数の初期値へのリセット

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。 以下の 2 つのオプションを使用できます。

- デフォルトの初期値へのリセット
- DEFINE DATA で定義した初期値へのリセット

注意:

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。

デフォルトの初期値へのリセット

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。

例：

```
DEFINE DATA LOCAL
1 #FIELDA (N3)  INIT <100>
1 #FIELDB (A20) INIT <'ABC'>
1 #FIELDC (I4)  INIT <5>
END-DEFINE
...
...
RESET #FIELDA                    /* resets field value to default initial value
...
```

DEFINE DATA で定義した初期値へのリセット

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。

例：

```
DEFINE DATA LOCAL
1 #FIELDA (N3)  INIT <100>
1 #FIELDB (A20) INIT <'ABC'>
1 #FIELDC (I4)  INIT <5>
END-DEFINE
...
RESET INITIAL #FIELDA #FIELDB #FIELDC /* resets field values to initial values as
defined in DEFINE DATA
...
```

21 フィールドの再定義

再定義は、フィールドのフォーマットを変更したり、単一フィールドを複数のセグメントに分割したりするために使用します。

DEFINE DATA ステートメントの REDEFINE オプションの使用

DEFINE DATA ステートメントの REDEFINE オプションは、単一のフィールド（ユーザー定義変数またはデータベースフィールド）を1つ以上の新しいフィールドとして再定義するために使用できます。 グループも再定義できます。

⚠ **重要:** ダイナミック変数は再定義には使用できません。

REDEFINE オプションは、フォーマットに関係なく、フィールドのバイト位置を左から右に再定義します。 バイト位置は、元のフィールドと再定義されたフィールド（複数可）の間で一致している必要があります。

再定義は元のフィールド定義の直後に指定する必要があります。

例 **1**：

以下の例では、データベースフィールド BIRTH を、3つの新しいユーザー定義変数として再定義しています。

```
DEFINE DATA LOCAL
01 EMPLOY-VIEW VIEW OF STAFFDDM
  02 NAME
  02 BIRTH
  02 REDEFINE BIRTH
     03 #BIRTH-YEAR  (N4)
     03 #BIRTH-MONTH (N2)
     03 #BIRTH-DAY   (N2)
END-DEFINE
   ...
```

例 **2**：

以下の例では、グループ #VAR2（フォーマット N および P の 2 つのユーザー定義変数で構成）を、フォーマット A の 1 つの変数として再定義しています。

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

FILLER *n*X 表記を使用して、再定義するフィールドに *n* バイトの充填バイト（使用しないセグメント）を定義できます。 末尾の充填バイトの定義は任意です。

例 **3**：

以下の例では、ユーザー定義変数 #FIELD を 3 つの新しいユーザー定義変数（それぞれフォーマット／長さが A2）として再定義しています。 FILLER 表記によって、元のフィールドの 3〜4、および 7〜10 バイト目を使用しないことが示されています。

```
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
...
```

再定義の使用方法を示すプログラム例

以下のプログラムは、再定義の使用方法を示しています。

```
** Example 'DDATAX01': DEFINE DATA
************************************************************************
DEFINE DATA LOCAL
01 VIEWEMP VIEW OF EMPLOYEES
  02 NAME
  02 FIRST-NAME
  02 SALARY   (1:1)
*
01 #PAY       (N9)
01 REDEFINE #PAY
  02 FILLER   3X
  02 #USD     (N3)
  02 #000     (N3)
END-DEFINE
*
READ (3) VIEWEMP BY NAME STARTING FROM 'JONES'
  MOVE SALARY (1) TO #PAY
  DISPLAY NAME FIRST-NAME #PAY #USD #000
END-READ
END
```

プログラム DDATAX01 の出力：

#PAY およびその再定義でできたフィールドがどのように表示されるかに注意してください。

```
Page      1                                                 04-11-11  14:15:54

        NAME              FIRST-NAME          #PAY    #USD #000
-------------------- -------------------- ---------- ---- ----

JONES                VIRGINIA                46000   46    0
JONES                MARSHA                  50000   50    0
JONES                ROBERT                  31000   31    0
```

22 配列

Naturalは配列の処理をサポートしています。配列は、複数次元のテーブル、つまり、単一の名前で識別される2つ以上の論理的に関連する要素です。 配列は、複数次元の単一データ要素、または連続する構造体や個別の要素を含む階層的なデータ構造で構成できます。

配列の定義

Natural では、1次元、2次元、または3次元の配列を使用できます。 配列は、独立した変数、より大きなデータ構造の一部、またはデータベースビューの一部です。

⚠ **重要:** ダイナミック変数は配列定義には使用できません。

手順 22.1. 1 次元配列を定義するには

- フォーマットと長さの後に、スラッシュと 「インデックス表記」、つまり配列のオカレンス数を指定します。

  例えば、以下の1次元配列には、3つのオカレンス（各オカレンスのフォーマット／長さはA10）があります。

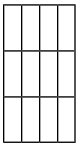
```
DEFINE DATA LOCAL
1 #ARRAY (A10/1:3)
END-DEFINE
...
```

手順 22.2. 2 次元配列を定義するには

- 両方の次元に対するインデックス表記を指定します。

```
DEFINE DATA LOCAL
1 #ARRAY (A10/1:3,1:4)
END-DEFINE
...
```

2 次元配列はテーブルとして表すことができます。 上記の例で定義されている配列は、3 つの 「行」 および 4 つの 「列」 から成るテーブルです。



配列の初期値

配列の 1 つ以上のオカレンスに初期値を割り当てるには、以下の例のように、「通常の」変数に対する初期値定義と同様に INIT 指定を使用します。

1 次元配列への初期値の割り当て

以下の例は、初期値が 1 次元配列にどのように割り当てられるかを説明しています。

- 初期値を 1 つのオカレンスに割り当てる場合、次を指定します。

```
1 #ARRAY (A1/1:3) INIT (2) <'A'>
```

  A は、2 番目のオカレンスに割り当てられます。

- 同じ初期値をすべてのオカレンスに割り当てるには、以下のように指定します。

```
1 #ARRAY (A1/1:3) INIT ALL <'A'>
```

  A は、すべてのオカレンスに割り当てられます。 または、次を指定できます。

```
1 #ARRAY (A1/1:3) INIT (*) <'A'>
```

- 同じ初期値を複数オカレンスの範囲に割り当てるには、以下のように指定します。

```
1 #ARRAY (A1/1:3) INIT (2:3) <'A'>
```

  A は、2〜3 番目のオカレンスに割り当てられます。

- 異なる初期値を全オカレンスに割り当てるには、以下のように指定します。

```
1 #ARRAY (A1/1:3) INIT <'A','B','C'>
```

A は最初のオカレンス、B は 2 番目のオカレンス、C は 3 番目のオカレンスにそれぞれ割り当てられます。

- 異なる初期値をいくつかのオカレンス（全オカレンスではなく）に割り当てるには、以下のように指定します。

```
1 #ARRAY (A1/1:3) INIT (1) <'A'> (3) <'C'>
```

A は最初のオカレンス、C は 3 番目のオカレンスに割り当てられますが、2 番目のオカレンスには値は割り当てられません。

または、次を指定できます。

```
1 #ARRAY (A1/1:3) INIT <'A',,'C'>
```

- オカレンス数より少ない数の初期値が指定されると、残ったオカレンスは空のままになります。

```
1 #ARRAY (A1/1:3) INIT <'A','B'>
```

A は最初のオカレンス、B は 2 番目のオカレンスに割り当てられますが、3 番目のオカレンスには値は割り当てられません。

2 次元配列への初期値の割り当て

このセクションでは、2次元配列に初期値を割り当てる方法について説明します。以下のトピックについて説明します。

- 異なる値の割り当て

前提条件

このセクションの例では、3 オカレンスの第 1 次元（「行」）と 4 オカレンスの第 2 次元（「列」）を持つ 2 次元配列を使用するものとします。

```
1 #ARRAY (A1/1:3,1:4)
```

縦方向：第 **1** 次元（**1:3**）、横方向：第 **2** 次元（**1:4**）

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)

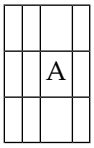
以下の最初の例では、*同じ*初期値を2次元配列のオカレンスにどのように割り当てるのかを説明します。2 番目の例では、*異なる*初期値をどのように割り当てるのかを説明します。

例では、*と∨の表記の使用に特に注意してください。これらの表記は両方とも、該当する次元のすべてのオカレンスを参照します。*は、その次元のすべてのオカレンスを*同じ*値で初期化することを示します。一方、∨は、その次元のすべてのオカレンスを*異なる*値で初期化することを示します。

同じ値の割り当て

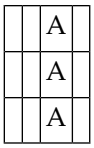
- 初期値を 1 つのオカレンスに割り当てる場合、次を指定します。

```
1 #ARRAY (A1/1:3,1:4) INIT (2,3) <'A'>
```



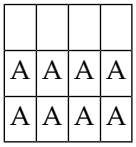
- 同じ初期値を第 2 次元の 1 オカレンス（第 1 次元の全オカレンス）に割り当てるには、以下のように指定します。

```
1 #ARRAY (A1/1:3,1:4) INIT (*,3) <'A'>
```



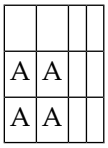
■ 同じ初期値を第1次元のオカレンス範囲（第2次元の全オカレンス）に割り当てるには、以下のように指定します。

```
1 #ARRAY (A1/1:3,1:4) INIT (2:3,*) <'A'>
```



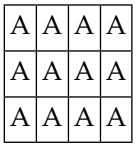
■ 同じ初期値を各次元のオカレンス範囲に割り当てるには、以下のように指定します。

```
1 #ARRAY (A1/1:3,1:4) INIT (2:3,1:2) <'A'>
```



■ 同じ初期値を全オカレンス（両方の次元）に割り当てるには、以下のように指定します。

```
1 #ARRAY (A1/1:3,1:4) INIT ALL <'A'>
```



または、次を指定できます。

```
1 #ARRAY (A1/1:3,1:4) INIT (*,*) <'A'>
```

異なる値の割り当て

■ `1 #ARRAY (A1/1:3,1:4) INIT (V,2) <'A','B','C'>`

	A		
	B		
	C		

■ `1 #ARRAY (A1/1:3,1:4) INIT (V,2:3) <'A','B','C'>`

	A	A	
	B	B	
	C	C	

■ `1 #ARRAY (A1/1:3,1:4) INIT (V,*) <'A','B','C'>`

A	A	A	A
B	B	B	B
C	C	C	C

■ `1 #ARRAY (A1/1:3,1:4) INIT (V,*) <'A',,'C'>`

A	A	A	A
C	C	C	C

■ `1 #ARRAY (A1/1:3,1:4) INIT (V,*) <'A','B'>`

A	A	A	A
B	B	B	B

■ `1 #ARRAY (A1/1:3,1:4) INIT (V,1) <'A','B','C'> (V,3) <'D','E','F'>`

A		D	
B		E	
C		F	

■ `1 #ARRAY (A1/1:3,1:4) INIT (3,V) <'A','B','C','D'>`

A	B	C	D

■ `1 #ARRAY (A1/1:3,1:4) INIT (*,V) <'A','B','C','D'>`

A	B	C	D
A	B	C	D
A	B	C	D

■ `1 #ARRAY (A1/1:3,1:4) INIT (2,1) <'A'> (*,2) <'B'> (3,3) <'C'> (3,4) <'D'>`

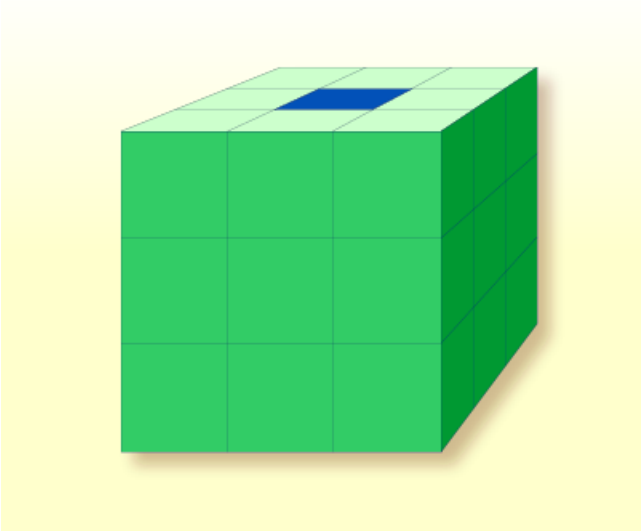
	B		
A	B		
	B	C	D

■ `1 #ARRAY (A1/1:3,1:4) INIT (2,1) <'A'> (V,2) <'B',C',D'> (3,3) <'E'> (3,4) <'F'>`

	B		
A	C		
	D	E	F

3 次元配列

3 次元配列は以下のように表すことができます。



上記の配列は、以下のように定義します（同時に、行 1、列 2、および面 2 の強調表示されたフィールドに初期値を割り当てます）。

```
DEFINE DATA LOCAL
1 #ARRAY2
  2 #ROW (1:4)
    3 #COLUMN (1:3)
      4 #PLANE (1:3)
        5 #FIELD2 (P3) INIT (1,2,2) <100>
END-DEFINE
...
```

データエリアエディタでローカルデータエリアとして定義すると、同じ配列は以下のように表示されます。

```
I T L Name                            F Leng Index/Init/EM/Name/Comment
- - - --------------------------------- - ---- ---------------------------------
    1 #ARRAY2
    2 #ROW                                   (1:4)
    3 #COLUMN                                (1:3)
    4 #PLANE                                 (1:3)
I   5 #FIELD2                           P    3
```

大きいデータ構造の一部としての配列

複数次元の配列を使用すると、COBOLまたはPL1の構造体に類似したデータ構造体を定義できます。

例：

```
DEFINE DATA LOCAL
1 #AREA
  2 #FIELD1 (A10)
  2 #GROUP1 (1:10)
    3 #FIELD2 (P2)
    3 #FIELD3 (N1/1:4)
END-DEFINE
...
```

この例では、データエリア #AREA の全体のサイズは以下のとおりです。

10 + (10 * (2 + (1 * 4))) バイト = 70 バイト

#FIELD1 は 10 バイトの長さの英数字です。 #GROUP1 は #AREA 内のサブエリアの名前で、2 つのフィールドから成り、10 オカレンスを持っています。 #FIELD2 は、パック型数値で長さは 2 です。 #FIELD3 は、4 オカレンスを持つ、#GROUP1 の 2 番目のフィールドで長さ 1 の数値です。

#FIELD3 の特定のオカレンスを参照するには、2 つのインデックスを使用する必要があります。最初のインデックスで #GROUP1 のオカレンスを指定し、2 番目のインデックスで #FIELD3 の特定のオカレンスを指定する必要があります。例えば、同じプログラムにおいて、後から ADD ステートメントで #FIELD3 を参照するには、以下のように指定します。

```
ADD 2 TO #FIELD3 (3,2)
```

データベース配列

Adabas は、マルチプルバリューフィールドおよびピリオディックグループの形で、データベース内の配列構造をサポートします。 これらについては、「*データベース配列*」を参照してください。

以下の例は、マルチプルバリューフィールドを含むビューの DEFINE DATA での定義を示しています。

```
DEFINE DATA LOCAL
1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (1:10) /* <--- MULTIPLE-VALUE FIELD
END-DEFINE
...
```

同じビューが、ローカルデータエリアでは以下のように表示されます。

```
I  T  L Name                             F Leng Index/Init/EM/Name/Comment
-  -  - --------------------------------- - ---- ---------------------------------
   V  1 EMPLOYEES-VIEW                             EMPLOYEES
     2 NAME                              A   20
   M  2 ADDRESS-LINE                     A   20 (1:10) /* MU-FIELD
```

インデックス表記での演算式の使用

配列のオカレンス範囲を表すために簡単な演算式をインデックスとして使用できます。

例：

MA (I:I+5)	値 I から I + 5 までの範囲のフィールド MA の値が参照されます。
MA (I+2:J-3)	値 I + 2 から J - 3 までの範囲のフィールド MA の値が参照されます。

添字の演算に使用できる演算子は、プラス（+）およびマイナス（-）のみです。

配列演算のサポート

配列演算のサポートには、配列レベル、行／列レベル、および個々の要素レベルでの演算が含まれています。

配列演算では、1つまたは2つのオペランドと受け取りフィールドの3番目の変数（任意）を使用した簡単な演算式のみを使用できます。

添字の範囲を定義する式には、演算子プラス（+）およびマイナス（-）のみを使用できます。

配列演算の例

以下の例では、次のフィールド定義を想定しています。

```
DEFINE DATA LOCAL
01 #A (N5/1:10,1:10)
01 #B (N5/1:10,1:10)
01 #C (N5)
END-DEFINE
...
```

1. `ADD #A(*,*) TO #B(*,*)`

   結果オペランドである配列 #B には、配列 #A と配列 #B の元の値を要素ごとに加算した結果が格納されます。

2. `ADD 4 TO #A(*,2)`

   配列 #A の 2 番目の列は、その元の値に 4 を加えた値で置き換えられます。

3. `ADD 2 TO #A(2,*)`

   配列 #A の 2 番目の行は、その元の値に 2 を加えた値で置き換えられます。

4. `ADD #A(2,*) TO #B(4,*)`

   配列 #A の 2 番目の行の値は、配列 #B の 4 番目の行に加算されます。

5. `ADD #A(2,*) TO #B(*,2)`

   これは不正な演算であるため、構文エラーが発生します。 行は行に、列は列にのみ加算できます。

6. `ADD #A(2,*) TO #C`

   配列 #A の 2 番目の行のすべての値は、スカラ値 #C に加算されます。

7. `ADD #A(2,5:7) TO #C`

   配列 #A の 2 番目の行の 5、6、7 番目の列の値がスカラ値 #C に加算されます。

23 X-array

通常の配列のフィールドを定義するときは、インデックスの範囲、つまり次元ごとのオカレンス数を正確に指定する必要があります。 ランタイムには配列のすべてのフィールドがデフォルトで存在するため、追加の割り当て操作を行うことなく、定義した各オカレンスにアクセスできます。 サイズのレイアウトは変更できません。したがって、フィールドのオカレンスを追加することも削除することもできません。

一方、必要なオカレンス数が開発時にわからないため、ランタイムに配列のフィールド数を柔軟に増減したい場合は、X-array（eXtensible array：拡張可能な配列）と呼ばれているものを使用します。

X-arrayでは、ランタイムにサイズ変更できるため、メモリをより効率よく管理できます。例えば、大量の配列オカレンスを短期間だけ使用してから、アプリケーションが配列を使用しなくなった時点でメモリを削減することができます。

定義

X-array とは、コンパイル時にオカレンス数を定義しない配列のことです。 X-array は、DEFINE DATA ステートメント内で、少なくとも 1 つの次元のインデックス範囲に、アスタリスク（*）を使用して定義します。インデックス定義内のアスタリスク文字（*）は、プログラムの実行時に特定の値を割り当てることができる、変更可能なインデックス範囲を表します。 上限または下限のいずれかのみを可変として定義できます。両方を定義することはできません。

X-array は、（固定の）配列を定義できる場所にならどこにでも、つまり、任意のレベルで、またはインデックス付きのグループとして定義できます。ただし、データベースビューのMU-/PE-フィールドへのアクセスには使用できません。 複数次元の配列では、定数と変更可能な境界を組み合わせて使用できます。

例：

```
DEFINE DATA LOCAL
1 #X-ARR1 (A5/1:*)          /* lower bound is fixed at 1, upper bound is variable
1 #X-ARR2 (A5/*)            /* shortcut for (A5/1:*)
1 #X-ARR3 (A5/*:100)        /* lower bound is variable, upper bound is fixed at 100
1 #X-ARR4 (A5/1:10,1:*)     /* 1st dimension has a fixed index range with (1:10)
END-DEFINE                  /* 2nd dimension has fixed lower bound 1 and variable
upper bound
```

X-array のストレージ管理

X-arrayのオカレンス数は、アクセスする前に明示的に割り当てる必要があります。次元のオカレンス数を増減するには、EXPAND、RESIZE、REDUCE の各ステートメントを使用します。

ただし、X-array の次元数（1、2、または 3 次元）は変更できません。

例：

```
DEFINE DATA LOCAL
1 #X-ARR(I4/10:*)
END-DEFINE
EXPAND ARRAY #X-ARR TO (10:10000)
/* #X-ARR(10) to #X-ARR(10000) are accessible
WRITE *LBOUND(#X-ARR)                  /* is 10
   *UBOUND(#X-ARR)                     /* is 10000
   *OCCURRENCE(#X-ARR)                 /* is 9991
#X-ARR(*) := 4711                      /* same as #X-ARR(10:10000) := 4711
/* resize array from current lower bound=10 to upper bound =1000
RESIZE ARRAY #X-ARR TO (*:1000)
/* #X-ARR(10) to #X-ARR(1000) are accessible
/* #X-ARR(1001) to #X-ARR(10000) are released
WRITE *LBOUND(#X-ARR)                  /* is 10
   *UBOUND(#X-ARR)                     /* is 1000
   *OCCURRENCE(#X-ARR)                 /* is 991
/* release all occurrences
REDUCE ARRAY #X-ARR TO  0
WRITE *OCCURRENCE(#X-ARR)              /* is 0
```

X-Group 配列のストレージ管理

X-group配列のオカレンス数を増減する場合、独立した次元と従属した次元を区別する必要があります。

直接指定されている（継承していない）X-group 配列または X-array の次元は、*独立*しています。

直接指定されていない（上位の配列から継承している）X-group 配列またはX-array の次元は、*従属*しています。

X-array の独立した次元のみ、EXPAND、RESIZE、REDUCE の各ステートメントで変更できます。次元の変更は、その次元を独立した次元として所有している X-group 配列の名前を使用して行う必要があります。

独立した次元と従属した次元の例：

```
DEFINE  DATA LOCAL
1 #X-GROUP-ARR1(1:*)                  /* (1:*)
  2 #X-ARR1    (I4)                   /* (1:*)
  2 #X-ARR2    (I4/2:*)               /* (1:*,2:*)
  2 #X-GROUP-ARR2                     /* (1:*)
    3 #X-ARR3  (I4)                   /* (1:*)
    3 #X-ARR4  (I4/3:*)               /* (1:*,3:*)
    3 #X-ARR5  (I4/4:*, 5:*)          /* (1:*,4:*,5:*)
END-DEFINE
```

以下の表は、上記のプログラム内の各次元が独立しているのか従属しているのかを示しています。

名前	従属した次元	独立した次元
#X-GROUP-ARR1		(1:*)
#X-ARR1	(1:*)	
#X-ARR2	(1:*)	(2:*)
#X-GROUP-ARR2	(1:*)	
#X-ARR3	(1:*)	
#X-ARR4	(1:*)	(3:*)
#X-ARR5	(1:*)	(4:*,5:*)

従属した次元に使用できる添字表記は、単一のアスタリスク（*）、アスタリスクで定義した範囲（*:*）、または定義済みの添字範囲のみです。

これは、従属した次元の境界は変更できないことを示しています。

従属した次元のオカレンス数は、対応するグループ配列を操作することによってのみ変更できます。

```
EXPAND ARRAY #X-GROUP-ARR1 TO (1:11)       /* #X-ARR1(1:11) are allocated
                                           /* #X-ARR3(1:11) are allocated
EXPAND ARRAY #X-ARR2 TO (*:*, 2:12)        /* #X-ARR2(1:11, 2:12) are allocated
EXPAND ARRAY #X-ARR2 TO (1:*, 2:12)        /* same as before
EXPAND ARRAY #X-ARR2 TO (*  , 2:12)        /* same as before
EXPAND ARRAY #X-ARR4 TO (*:*, 3:13)        /* #X-ARR4(1:11, 3:13) are allocated
EXPAND ARRAY #X-ARR5 TO (*:*, 4:14, 5:15)  /* #X-ARR5(1:11, 4:14, 5:15) are allocated
```

EXPAND ステートメントは、任意の順序で記述できます。

従属した次元しか配列にないため、以下のような EXPAND ステートメントは使用できません。

```
EXPAND ARRAY #X-ARR1 TO ...
EXPAND ARRAY #X-GROUP-ARR2 TO ...
EXPAND ARRAY #X-ARR3 TO ...
```

X-array の参照

X-array のオカレンス数は、アクセスする前に EXPAND ステートメントまたは RESIZE ステートメントを使用して、明示的に割り当てる必要があります。

一般的なルールとして、存在しないX-arrayのオカレンスにアクセスしようとするとランタイムエラーが発生します。 ただし、ステートメントの中には、全体範囲表記（#X-ARR(*) など）を使用して X-array のすべてのオカレンスを参照する場合は、実体を持たない X-array フィールドにアクセスしてもエラーにならないものがあります。 これは、以下に対して適用されます。

- CALL ステートメント内で使用されているパラメータ
- オプションパラメータとして定義されている場合は、CALLNAT、PERFORM、OPEN DIALOG の各ステートメント内で使用されているパラメータ
- COMPRESS ステートメント内で使用されているソースフィールド
- PRINT ステートメント内で指定されている出力フィールド
- RESET ステートメント内で参照されているフィールド

これらのステートメントの 1 つを使用して実体を持たない X-array のオカレンスを個別に参照すると、対応するエラーメッセージが発行されます。

例：

```
DEFINE DATA LOCAL
1 #X-ARR (A10/1:*) /* X-array only defined, but not allocated
END-DEFINE
RESET #X-ARR(*)    /* no error, because complete field referenced with (*)
RESET #X-ARR(1:3)  /* runtime error, because individual occurrences (1:3) are
referenced
END
```

配列参照のアスタリスク（*）表記は、次元のすべての範囲を表します。配列がX-arrayの場合、アスタリスクは、現在割り当てられている下限および上限のインデックス範囲を表します。この下限および上限は、*LBOUND および *UBOUND によって確認できます。

X-array を使用したパラメータ引き渡し

X-array をパラメータとして使用すると、以下の整合性チェックで定数の配列のように扱われます。

- フォーマット
- 長さ
- 次元
- オカレンス数

また、X-array パラメータは、ステートメント RESIZE、REDUCE、または EXPAND を使用することにより、オカレンス数を変更することもできます。X-array パラメータのサイズ変更が可能かどうかは、以下の 3 つの要素によって決まります。

- 使用するパラメータ引き渡しのタイプ（参照渡しまたは値渡し）
- 呼び出し元または X-array パラメータの定義
- 引き渡す X-array の範囲のタイプ（全体または一部）

以下の表は、X-array パラメータに EXPAND、RESIZE、REDUCE の各ステートメントを適用できる条件を示しています。

値渡しの例

呼び出し元	パラメータ		
	スタティック	変数（1:V）	X-array
スタティック	×	×	○
X-array の一部。例：CALLNAT...#XA(1:5)	×	×	○
X-array 全体。例：CALLNAT...#XA(*)	×	×	○

参照渡し／値渡し（結果）

呼び出し元	パラメータ

	スタティック	変数（1:V）	固定の下限を持つX-array。例： `DEFINE DATA PARAMETER 1 #PX (A10/1:*)`	固定の上限を持つX-array。例： `DEFINE DATA PARAMETER 1 #PX (A10/*:1)`
スタティック	×	×	×	×
X-array の一部。例： `CALLNAT...#XA(1:5)`	×	×	×	×

固定の下限を持つ X-array の全体。例：
 	
`DEFINE DATA LOCAL`
`1 #XA(A10/1:*)`
`...`
`CALLNAT...#XA(*)`
	×	×	○	×

固定の上限を持つ X-array の全体。例：
 	
`DEFINE DATA LOCAL`
`1 #XA(A10/*:1)`
`...`
`CALLNAT...#XA(*)`
	×	×	×	○

X-group 配列を使用したパラメータ引き渡し

X-group配列の宣言は、グループの各要素が同じ上限および下限を持つことを暗黙的に意味します。 したがって、X-group配列フィールドの従属した次元のオカレンス数は、RESIZE、REDUCE、EXPAND の各ステートメントで X-group 配列のグループ名を指定したときにのみ変更できます（前述の「*X-Group 配列のストレージ管理*」を参照）。

X-group 配列のメンバは、パラメータデータエリアに定義されている X-group 配列にパラメータとして引き渡すことができます。 呼び出す側と呼び出される側のグループ構造は必ずしも同一である必要はありません。 呼び出した側の X-group 配列に矛盾がない限り、呼び出された側は RESIZE、REDUCE および EXPAND を実行できます。

パラメータとして引き渡す **X-group** 配列の要素の例：

プログラム：

```
DEFINE DATA LOCAL
1 #X-GROUP-ARR1(1:*)                    /* (1:*)
   2 #X-ARR1 (I4)                       /* (1:*)
   2 #X-ARR2 (I4)                       /* (1:*)
1 #X-GROUP-ARR2(1:*)                    /* (1:*)
   2 #X-ARR3 (I4)                       /* (1:*)
   2 #X-ARR4 (I4)                       /* (1:*)
END-DEFINE
...
CALLNAT ... #X-ARR1(*) #X-ARR4(*)
...
END
```

サブプログラム：

```
DEFINE DATA PARAMETER
1 #X-GROUP-ARR(1:*)                     /* (1:*)
   2 #X-PAR1 (I4)                       /* (1:*)
   2 #X-PAR2 (I4)                       /* (1:*)
END-DEFINE
...
RESIZE ARRAY #X-GROUP-ARR to (1:5)
...
END
```

サブプログラムの RESIZE ステートメントは実行できません。 プログラムの X-group 配列で定義されているフィールドのオカレンス数と矛盾しています。

ダイナミック変数の X-array

ダイナミック変数の X-array の割り当てでは、最初に EXPAND ステートメントを使用してオカレンス数を指定し、後でそのオカレンス数に値を割り当てることによって変更できます。

例：

```
DEFINE DATA LOCAL
 1 #X-ARRAY(A/1:*)  DYNAMIC
END-DEFINE
EXPAND  ARRAY   #X-ARRAY TO (1:10)
  /* allocate #X-ARRAY(1) to #X-ARRAY(10) with zero length.
  /* *LENGTH(#X-ARRAY(1:10)) is zero
#X-ARRAY(*) := 'abc'
  /* #X-ARRAY(1:10) contains 'abc',
  /* *LENGTH(#X-ARRAY(1:10)) is 3
EXPAND  ARRAY   #X-ARRAY TO (1:20)
  /* allocate #X-ARRAY(11) to #X-ARRAY(20) with zero length
  /* *LENGTH(#X-ARRAY(11:20)) is zero
#X-ARRAY(11:20) := 'def'
  /* #X-ARRAY(11:20) contains 'def'
  /* *LENGTH(#X-ARRAY(11:20)) is 3
```

配列の上限および下限

システム変数*LBOUNDおよび*UBOUNDには、指定した次元（1、2、または3次元。複数指定可）に対する配列の、その時点での下限および上限が設定されます。

X-array にオカレンスが割り当てられていない場合、*LBOUND または *UBOUND は未定義のため、アクセスするとランタイムエラーが発生します。 ランタイムエラーを回避するには、*LBOUND または *UBOUND を評価する前に *OCCURRENCE を使用して、オカレンス数がゼロかどうかを確認します。

例：

```
IF *OCCURRENCE (#A) NE 0  AND  *UBOUND(#A) < 100 THEN ...
```

24 データベース内のデータへのアクセス

ここでは、Naturalを使用してデータベースのデータにアクセスするときのさまざまな面について説明します。

- **Natural** およびデータベースへのアクセス
- **Adabas** データベースのデータへのアクセス
- **SQL** データベースのデータへのアクセス
- **VSAM** データベースのデータへのアクセス
- **DL/I** データベースのデータへのアクセス

以下の項目も参照してください。

- 各種のデータベース管理システムのデータベースインターフェイスの概要については、「*DBMS インターフェイス*」を参照してください。
- Natural からのデータベースアクセス処理に影響を与える Natural プロファイルパラメータの概要については、「*プロファイルパラメータの概要*」の「データベースへのアクセス」を参照してください。

25 Natural およびデータベースへのアクセス

この章では、異なるタイプのデータベース管理システムおよびファイル管理システムに対して Natural で提供される機能の概要について説明します。

Natural でサポートされるデータベース管理システム

Natural では、次のタイプのデータベース管理システム（DBMS）に対して特定のデータベースインターフェイスが提供されています。

- ネスト構造のリレーショナル DBMS（Adabas）
- SQL タイプの DBMS（DB2（SQL/DS）、Oracle、Sybase、Informix、MS SQL Server）
- ファイルシステム（VSAM）
- DL/I

以下では次のトピックについて説明します。

Adabas

Natural は、統合された Adabas インターフェイスを使用することによって、ローカルマシンまたはリモートコンピュータ上の Adabas データベースにアクセスすることができます。 リモートアクセスの場合は、Entire Net-Work などのルーティングおよび通信用の追加のソフトウェアが必要です。 いずれの場合も、Adabas データベースを実行しているホストマシンのタイプは、Natural ユーザーに透過的です。

SQL データベース

Natural for DB2 では、DB2 および SQL/DS データベース管理システムにアクセスするためのステートメントの一般セットが提供されます。 詳細については、『*データベース管理システムインターフェイス*』ドキュメントの次のアドオン製品に関する説明を参照してください。

- *Natural for DB2*
- *Natural for SQL/DS*
- *Natural SQL Gateway*

VSAM

Natural ユーザーは、VSAM への Natural インターフェイスを使用して、VSAM ファイルに保存されているデータにアクセスすることができます。Naturalのステートメントおよびシステム変数を VSAM で使用するときの詳細および特別な考慮事項については、『*データベース管理システムインターフェイス*』ドキュメントの「*Natural for VSAM*」を参照してください。

DL/I

Natural ユーザーは Natural for DL/I を使用して、DL/I データベースに保存されているデータにアクセスし、データを更新することができます。処理はバッチモードで実行することも、TP モニタの CICS または IMS/TM の制御下で実行することもできます。DL/I データベースは、Natural ではファイルの集まりとして表現され、各ファイルが 1 つのデータベースセグメントタイプを表します。 それぞれのファイル、つまりセグメントタイプには、Natural FDIC システムファイルで生成され、保存される DDM を関連付ける必要があります。

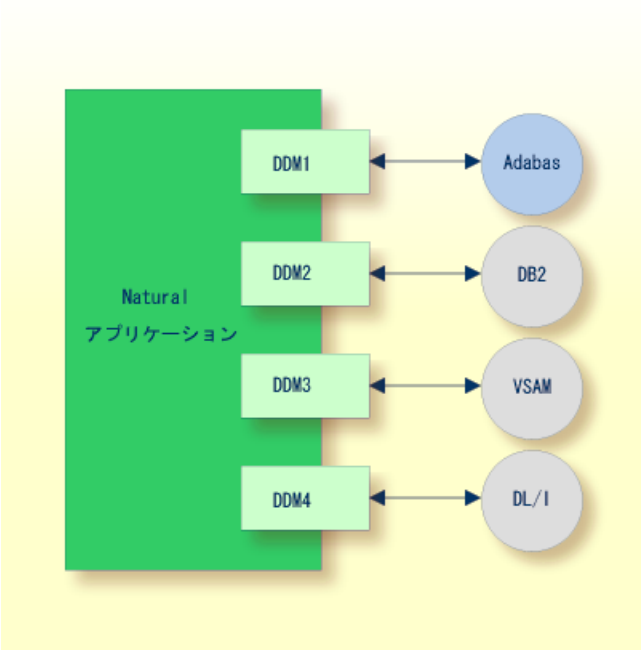
DL/I データベースへのアクセスに使用する Natural ステートメントは、Natural 言語が提供されるステートメントのサブセットになります。 DL/I データベースにアクセスするための新しいステートメントは不要です。

詳細については、『*データベース管理システムインターフェイス*』ドキュメントの「*Natural for DL/I*」を参照してください。

データ定義モジュールを使用したアクセス

異なるデータベース管理システムへのアクセスを便利で透過的なものにするために、Natural では「データ定義モジュール」（DDM）という特殊なオブジェクトが使用されます。 この DDM によって、Naturalのデータ構造と、使用するデータベースシステムのデータ構造との間の接続が確立されます。該当するデータベース構造には、SQL データベースのテーブルまたは Adabas データベースのファイルなどがあります。したがって、Naturalアプリケーションからアクセスするデータベースの実際の構造は DDM によって隠されます。 DDM は、Natural DDM エディタを使用して作成します。

Natural は、特定のデータベースシステムの特定のデータ構造を表す DDM への参照を使用して、単一アプリケーション内から複数のデータベースタイプ（Adabas、DB2、VSAM、DL/I）にアクセスすることができます。 次の図は、異なるタイプのデータベースに接続する 1 つのアプリケーションを示しています。



Natural のデータ操作言語

Natural には、サポートするすべてのデータベースシステムに対して FIND、READ、STORE、DELETE などの同じ言語ステートメントを使用して Natural アプリケーションからアクセスできるようにするデータ操作言語（DML）が組み込まれています。 アクセスするデータベースのタイプがわからなくても、Natural アプリケーションでこれらのステートメントを使用することができます。

Natural は、データベースシステムの実際のタイプをコンフィグレーションファイルから特定し、DML ステートメントをデータベース固有のコマンドに変換します。つまり、Adabas に対してはダイレクトコマンド、SQL データベースに対しては SQL ステートメント文字列とホスト変数構造を生成します。

一部の Natural DML ステートメントでは、すべてのデータベースタイプにはサポートされていない機能が提供されるため、このような機能の使用は特定データベースシステムのみに制限されます。 ステートメントドキュメントで、該当するデータベース固有の考慮事項を参照してください。

Natural の特殊な SQL ステートメント

「通常の」 Natural DML ステートメントに加えて、Natural では SQL データベースシステムと連携した、より限定して使用するための一連の SQL ステートメントが提供されます。『ステートメント』ドキュメントの「*SQL ステートメントの概要*」を参照してください。

ストアドプロシージャを使用するためのフレキシブル SQL や各種機能も SQL コマンドのセットに含まれています。 これらのステートメントは、SQL データベースへのアクセスにのみ使用可能です。Adabas をはじめとする SQL 以外のデータベースには有効ではありません。

26 Adabas データベースのデータへのアクセス

この章では、Natural を使用して Adabas データベースのデータにアクセスするときのさまざまな面について説明します。

Natural を Adabas と使用するときに適用される Natural プロファイルパラメータの概要を説明する『*オペレーション*』ドキュメントの「*Adabas での Natural*」も参照してください。

データ定義モジュール - DDM

Naturalがデータベースファイルにアクセスできるようにするには、物理データベースファイルの論理定義が必要です。このような論理ファイル定義は、データ定義モジュール（DDM）と呼ばれます。

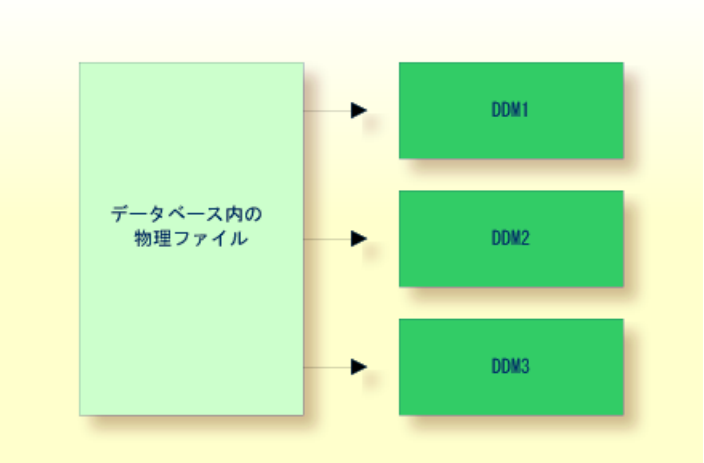
このセクションでは、次のトピックについて説明します。

- データ定義モジュールの使用
- DDM の管理
- DDM のリスト／表示

データ定義モジュールの使用

データ定義モジュールには、ファイルの個々のフィールドについての情報が含まれています。この情報は、Natural プログラムでこれらのフィールドを使用するために必要です。DDM は、物理データベースファイルの論理ビューを構成しています。

データベースの各物理ファイルに、1 つ以上の DDM を定義できます。各 DDM には、1 つ以上のデータビューを定義できます。ステートメントドキュメントの DEFINE DATA の「*ビューの定義*」を参照してください。



DDM は、Natural 管理者が Predict（Predict を使用できない場合は、対応する Natural 機能）を使用して定義します。

DDM の管理

システムコマンド SYSDDM を使用して、SYSDDM ユーティリティを呼び出します。 SYSDDM ユーティリティを使用すると、Naturalのデータ定義モジュールの作成および管理に必要なすべての機能を実行できます。

SYSDDMユーティリティの詳細については、『エディタ』ドキュメントの「*SYSDDM* ユーティリティ」を参照してください。

各データベースフィールドについて、DDMにはデータベース内部のフィールド名の他に、Naturalプログラムで使用されるフィールドの名前である「外部」フィールド名も含まれています。また、フィールドのフォーマットと長さ、およびフィールドが DISPLAY または WRITE ステートメントで出力されるときに使われる各種の指定（列見出し、編集マスクなど）もDDMに定義されます。

DDM に定義されるフィールド属性については、『エディタ』ドキュメントの「*SYSDDM* ユーティリティ」セクションの「*DDM エディタ画面の使用*」を参照してください。

DDM のリスト／表示

希望する DDM の名前がわからない場合は、システムコマンド LIST DDM を使用して、現在のライブラリで使用できる既存のすべての DDM のリストを取得できます。 このリストから、表示する DDM を選択できます。

名前がわかっている DDM を表示するには、システムコマンド LIST DDM *ddm-name* を使用します。

例：

```
LIST DDM EMPLOYEES
```

DDMに定義されているすべてのフィールドのリストが、各フィールドの情報とともに表示されます。DDMに定義されるフィールド属性については、『エディタ』ドキュメントの「*SYSDDM* ユーティリティ」を参照してください。

データベース配列

Adabas は、マルチプルバリューフィールドおよびピリオディックグループの形で、データベース内の配列構造をサポートします。

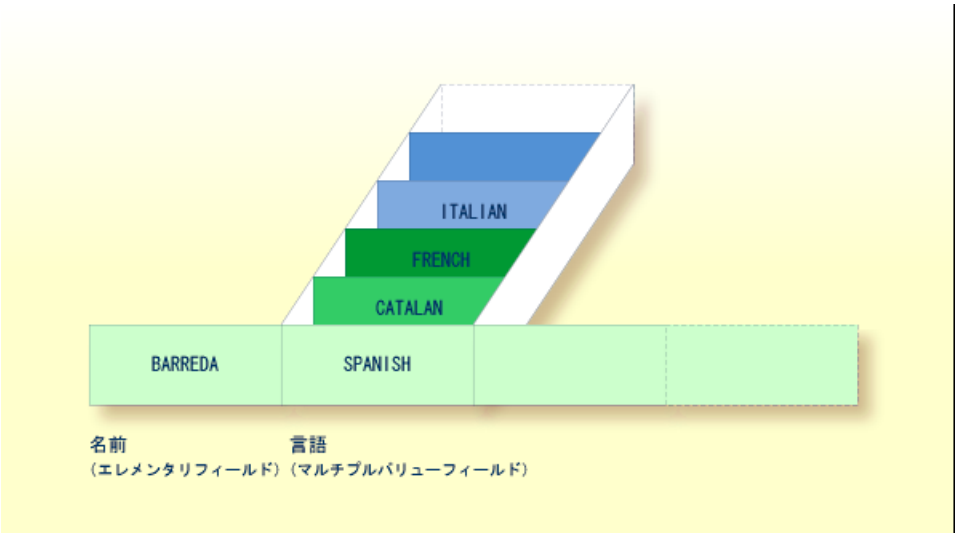
このセクションでは、次のトピックについて説明します。

- 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
- 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参照

マルチプルバリューフィールド

マルチプルバリューフィールドは、任意のレコード内に複数の値を持つことができるフィールドです。値の数は65534以下ですが、Adabasバージョンおよび FDT の定義に応じて異なります。

例：



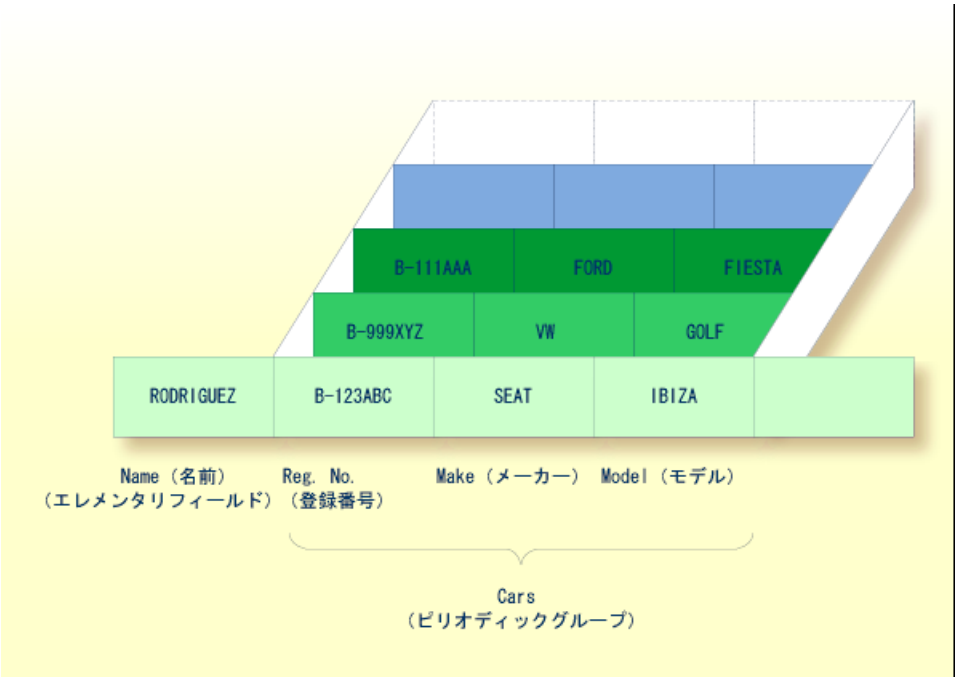
上記の図を EMPLOYEES ファイルのレコードと仮定すると、最初のフィールド（Name）はエレメンタリフィールドであり、1つの値、つまり従業員の名前のみを含めることができます。これに対して2つ目のフィールド（Languages）には、従業員が話す言語が含まれています。従業員は複数の言語を話せる可能性があるので、マルチプルバリューフィールドになっています。

ピリオディックグループ

ピリオディックグループは、任意のレコード内で複数のオカレンスを持つことができるエレメンタリフィールドまたはマルチプルバリューフィールドのグループです。オカレンスの数は65534以下ですが、Adabas バージョンおよび FDT の定義に応じて異なります。

マルチプルバリューフィールドに含まれる各値は、一般に 「オカレンス」 と呼ばれます。オカレンスの数は、フィールドに含まれる値の数であり、特定のオカレンスは特定の値を表します。同様に、ピリオディックグループでは、オカレンスは複数の値の集まりを表します。

例：



上の図が車両ファイル内のレコードであると仮定すると、最初のフィールド（Name）は人物の名前を含むエレメンタリフィールドです。「Cars」は、その人物が所有する自動車を含むピリオディックグループです。 ピリオディックグループは、各自動車の登録番号、メーカー、およびモデルの 3 つのフィールドで構成されています。 Cars の各オカレンスには 1 台の車に関する値が含まれます。

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参照

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の1つ以上のオカレンスを参照するには、フィールド名の後に 「インデックス表記」 を指定します。

例：

次の例では、上述の例のマルチプルバリューフィールド `LANGUAGES` とピリオディックグループ `CARS` を使用します。

マルチプルバリューフィールド LANGUAGES の各値は、次のように参照できます。

例	Explanation
LANGUAGES (1)	最初の値（SPANISH）を参照します。
LANGUAGES (X)	変数 X の値によって、参照する値が決まります。
LANGUAGES (1:3)	最初の 3 つの値（SPANISH、CATALAN、および FRENCH）を参照します。
LANGUAGES (6:10)	6 番目から 10 番目の値を参照します。
LANGUAGES (X:Y)	変数 X と変数 Y の値によって、参照する値が決まります。

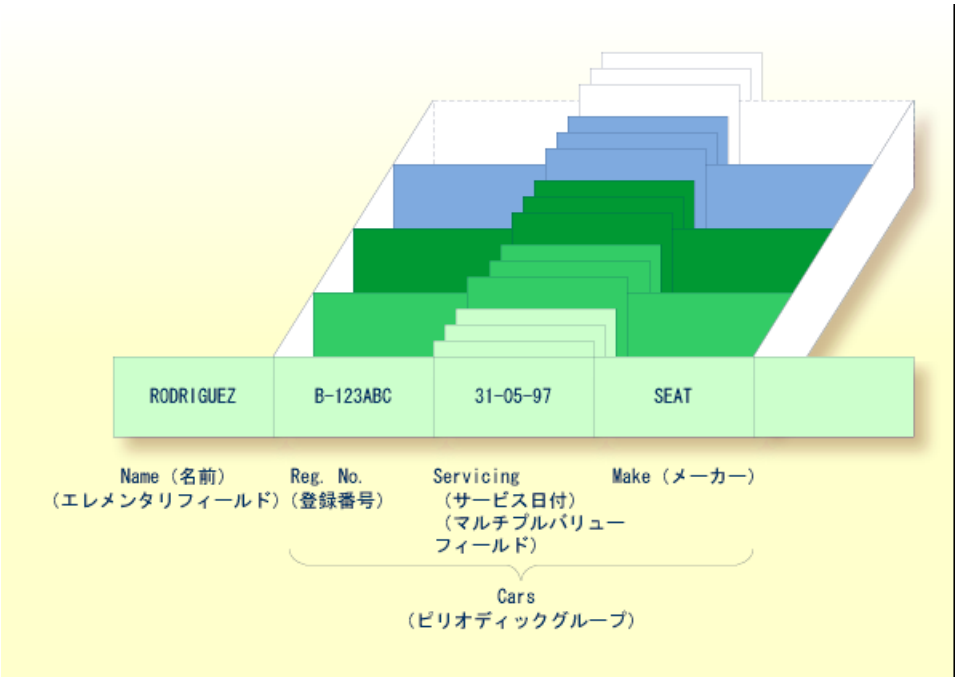
ピリオディックグループ CARS のさまざまなオカレンスも同様の方法で参照できます。

例	Explanation
CARS (1)	最初のオカレンス（B-123ABC/SEAT/IBIZA）を参照します。
CARS (X)	変数 X の値によって、参照するオカレンスが決まります。
CARS (1:2)	最初の 2 つのオカレンス（B-123ABC/SEAT/IBIZA と B-999XYZ/VW/GOLF）を参照します。
CARS (4:7)	4 番目から 7 番目のオカレンスを参照します。
CARS (X:Y)	変数 X と変数 Y の値によって、参照するオカレンスが決まります。

ピリオディックグループ内のマルチプルバリューフィールド

Adabas 配列には、2 次元まで、つまり 1 つのピリオディックグループ内に 1 つのマルチプルバリューフィールドを含めることができます。

例：



上の図が車両ファイル内のレコードであると仮定すると、最初のフィールド（Name）は人物の名前を含むエレメンタリフィールドです。「Cars」は、その人物が所有する自動車を含むピリオディックグループです。 ピリオディックグループは、各自動車の登録番号、サービス日付、およびメーカーの3つのフィールドで構成されています。ピリオディックグループCars内のフィールドServicingはマルチプルバリューフィールドであり、各車の異なるサービス日付が含まれています。

ピリオディックグループ内のマルチプルバリューフィールドの参照

ピリオディックグループ内のマルチプルバリューフィールドの1つ以上のオカレンスを参照するには、フィールド名の後に 「インデックス表記」 を指定します。

例：

次の例では、上述の例のピリオディックグループ CARS 内のマルチプルバリューフィールド SERVICING を使用します。マルチプルバリューフィールドの各値は、次のように参照できます。

例	Explanation
SERVICING (1,1)	CARS の最初のオカレンスにある SERVICING の最初の値（31-05-97）を参照します。
SERVICING (1:5,1)	CARS の最初の5つのオカレンスにある SERVICING の最初の値を参照します。
SERVICING (1:5,1:10)	CARS の最初の 5 つのオカレンスにある SERVICING の最初の 10 個の値を参照します。

データベース配列の内部カウントの参照

レコードに値またはオカレンスがいくつ存在するかが不明な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参照が必要になることがあります。 Adabas では、各マルチプルバリューフィールドの値の数、および各ピリオディックグループのオカレンスの数の内部カウントが保持されています。 このカウントは、READ ステートメントでフィールド名の直前に C* を指定することによって読み込むことができます。

カウントは、フォーマット／長さ N3 で返されます。 詳細については、「*データベース配列の内部カウントの参照*」を参照してください。

例	Explanation
C*LANGUAGES	マルチプルバリューフィールド LANGUAGES の値の数を返します。
C*CARS	ピリオディックグループ CARS のオカレンスの数を返します。
C*SERVICING (1)	ピリオディックグループの最初のオカレンスにあるマルチプルバリューフィールド SERVICING の値の数を返します。SERVICING はピリオディックグループ内のマルチプルバリューフィールドであると仮定しています。

DEFINE DATA ビュー

Natural プログラムでデータベースフィールドを使用できるようにするには、ビューでフィールドを指定する必要があります。

このセクションでは、次のトピックについて説明します。

- データベースビューの使用

データベースビューの使用

Naturalプログラムでデータベースフィールドを使用できるようにするには、ビューでフィールドを指定する必要があります。

ビューには次の内容を指定します。

- フィールドの取得元となるデータ定義モジュール（DDM）の名前。
- データベースフィールド自体の名前。つまり、データベース内部のショートネームではなく、ロングネーム。

データベースビューの定義

データベースビューは、次のいずれかの場所に定義します。

- プログラムの `DEFINE DATA` ステートメント内部
- プログラム外部のローカルデータエリア（LDA）またはグローバルデータエリア（GDA）。`DEFINE DATA` ステートメントを使用して、該当するデータエリアを参照します。詳細については、「*フィールドの定義*」セクションを参照してください。

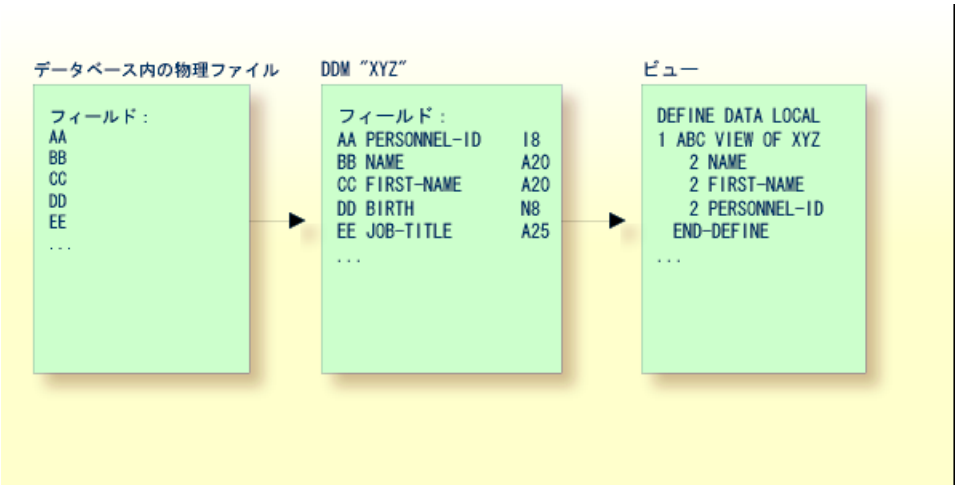
レベル 1 では、ビュー名を次のように指定します。

```
1 view-name VIEW OF ddm-name
```

`view-name` はビューに選択した名前、*`ddm-name`* はビューに指定されたフィールドの取得元となる DDM の名前です。

レベル 2 では、DDM のデータベースフィールドの名前を指定します。

次の図に示すように、ビューの名前は `ABC` で、DDM `XYZ` から取得したフィールド `NAME`、`FIRST-NAME`、および `PERSONNEL-ID` で構成されています。



データベースフィールドのフォーマットと長さは、基礎となるDDMですでに定義されているため、ビューに指定する必要はありません。

ビューは DDM 全体を含めたり、そのサブセットのみを含めたりすることができます。 ビューのフィールドの順番を、基礎となる DDM と同じにする必要はありません。

ビュー名は、アクセスするデータベースを決定するためにデータベースアクセスステートメントで使用されます。「データベースアクセスのステートメント」を参照してください。

データベースアクセスのステートメント

データベースからデータを読み込むには、次のステートメントを使用できます。

ステートメント	説明
READ	指定した順番でデータベースからレコードの範囲を選択します。
FIND	指定した検索条件に一致するレコードをデータベースから選択します。
HISTOGRAM	1 つのデータベースフィールドの値のみを読み込みます。または、指定した検索条件に一致するレコードの数を決定します。

READ ステートメント

次のトピックについて説明します。

READ ステートメントの使用

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。 レコードがデータベースから読まれる順番は次のとおりです。

- データベースに物理的に保存されている順番（READ IN PHYSICAL SEQUENCE）
- Adabas 内部シーケンス番号の順番（READ BY ISN）
- ディスクリプタフィールドの値の順番（READ IN LOGICAL SEQUENCE）

このドキュメントでは、READ IN LOGICAL SEQUENCE のみを取り上げます。これは最も頻繁に使用される形の READ ステートメントです。

他の 2 つのオプションの詳細については、『ステートメント』ドキュメントの READ ステートメントの説明を参照してください。

READ ステートメントの基本構文

READ ステートメントの基本構文は次のとおりです。

```
READ view IN LOGICAL SEQUENCE BY descriptor
```

または、以下のように短くすることができます。

```
READ view LOGICAL BY descriptor
```

各項目の意味を次に示します。

view	DEFINE DATA ステートメントで定義されるビューの名前（「*DEFINE DATA ビュー*」を参照）。
descriptor	そのビューで定義されるデータベースフィールドの名前。 このフィールドの値によって、データベースから読み込まれるレコードの順番が決まります。

ディスクリプタを指定する場合は、キーワード LOGICAL を指定する必要はありません。

```
READ view BY descriptor
```

ディスクリプタを指定しない場合は、**DDM** の デフォルト順 にデフォルトディスクリプタとして定義されたフィールドの値の順番でレコードが読み込まれます。 ただし、ディスクリプタを指定しない場合は、次のようにキーワード LOGICAL を指定する必要があります。

```
READ view LOGICAL
```

READ ステートメントの例

```
** Example 'READX01': READ
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
  2 JOB-TITLE
END-DEFINE
*
READ (6) MYVIEW BY NAME
  DISPLAY NAME PERSONNEL-ID JOB-TITLE
END-READ
END
```

プログラム READX01 の出力：

上記の例の READ ステートメントでは、EMPLOYEES ファイルのレコードが各従業員の姓のアルファベット順に読み込まれます。

プログラムによって次のような出力が作成され、各従業員の情報がそれぞれの姓のアルファベット順に表示されます。

```
Page      1                                                  04-11-11  14:15:54

        NAME          PERSONNEL          CURRENT
                         ID             POSITION
-------------------- --------- ------------------------

ABELLAN              60008339  MAQUINISTA
ACHIESON             30000231  DATA BASE ADMINISTRATOR
ADAM                 50005800  CHEF DE SERVICE
ADKINSON             20008800  PROGRAMMER
ADKINSON             20009800  DBA
ADKINSON             2001100
```

従業員を生年月日順にリストするレポートを作成するためにレコードを読み込む場合の適切な READ ステートメントは次のようになります。

```
READ MYVIEW BY BIRTH
```

指定できるのは、基礎となる **DDM** で「ディスクリプタ」として定義されているフィールドのみです。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、またはノンディスクリプタの場合もあります。

読み込むレコード数の制限

上記のプログラム例で示されているように、キーワード READ の後にカッコで囲んだ数字を次のように指定することによって、読み込まれるレコード数を制限できます。

```
READ (6) MYVIEW BY NAME
```

上記の例では、READ ステートメントで 6 件を超えるレコードは読み込まれなくなります。

リミット表記がない場合、上記の READ ステートメントによって、EMPLOYEES ファイルのすべてのレコードが姓の順に A から Z まで読み込まれます。

STARTING／ENDING 節

READ ステートメントでは、ディスクリプタフィールドの*値*に基づいてレコードの選択を限定することもできます。BY 節または WITH 節で EQUAL TO/STARTING FROM オプションを設定することによって、読み込みを開始する値を指定できます。THRU/ENDING AT オプションを追加して、読み込みを終了する値を論理順で指定することもできます。

例えば、TRAINEE で開始して Z まで継続する職種の順番で従業員をリストするには、次のいずれかのステートメントを使用します。

```
READ MYVIEW WITH JOB-TITLE = 'TRAINEE'
READ MYVIEW WITH JOB-TITLE STARTING from 'TRAINEE'
READ MYVIEW BY JOB-TITLE = 'TRAINEE'
READ MYVIEW BY JOB-TITLE STARTING from 'TRAINEE'
```

等号（=）または STARTING FROM オプションの右側の値はアポストロフィで囲む必要があります。 値が数値の場合は、このテキスト表記は不要です。

BY オプションを使用するときは、WITH オプションを使用できません。この逆も同様です。

読み込む一連のレコードは、THRU 節または ENDING AT 節で終了制限を追加することによって、より厳密に指定できます。

職種が TRAINEE のレコードだけを読み込むには、次のように指定します。

```
READ MYVIEW BY JOB-TITLE STARTING from 'TRAINEE' THRU 'TRAINEE'
READ MYVIEW WITH JOB-TITLE EQUAL TO 'TRAINEE'
                           ENDING AT 'TRAINEE'
```

職種が A または B で始まるレコードだけを読み込むには、次のように指定します。

```
READ MYVIEW BY JOB-TITLE = 'A' THRU 'C'
READ MYVIEW WITH JOB-TITLE STARTING from 'A' ENDING AT 'C'
```

値は、THRU/ENDING AT の後に指定された値まで、その値を含めて読み込まれます。 上記の 2 つの例では、職種が A または B で始まる全レコードが読み込まれます。職種 C があれば、これも読み込まれますが、次に高い値である CA は読み込まれません。

WHERE 節

WHERE 節を使用して、読み込むレコードをさらに限定することができます。

例えば、米国通貨で給与が支払われ、職種が TRAINEE で始まる従業員のみを必要とする場合は、次のように指定します。

```
READ MYVIEW WITH JOB-TITLE = 'TRAINEE'
            WHERE CURR-CODE = 'USD'
```

WHERE 節は、次のように BY 節とともに使用することもできます。

```
READ MYVIEW BY NAME
            WHERE SALARY = 20000
```

WHERE 節は、次の 2 つの点で BY/WITH 節と異なります。

- WHERE 節に指定するフィールドがディスクリプタである必要はありません。
- WHERE オプションに続く式は論理条件です。

WHERE 節では次の論理演算子が有効です。

EQUAL	EQ	=
NOT EQUAL TO	NE	¬=
LESS THAN	LT	<
LESS THAN OR EQUAL TO	LE	<=
GREATER THAN	GT	>
GREATER THAN OR EQUAL TO	GE	>=

次のプログラムは、STARTING FROM、ENDING AT、および WHERE の各節の使い方を説明するものです。

```
** Example 'READX02': READ  (with STARTING, ENDING and WHERE clause)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 INCOME     (1:2)
    3 CURR-CODE
    3 SALARY
    3 BONUS    (1:1)
END-DEFINE
*
READ (3) MYVIEW WITH  JOB-TITLE
STARTING FROM 'TRAINEE' ENDING AT 'TRAINEE'
                WHERE CURR-CODE (*) = 'USD'
```

```
  DISPLAY NOTITLE NAME / JOB-TITLE 5X INCOME (1:2)
  SKIP 1
END-READ
END
```

プログラム READX02 の出力：

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

SENKO                         USD            23000          0
TRAINEE                       USD            21800          0

BANGART                       USD            25000          0
TRAINEE                       USD            23000          0

LINCOLN                       USD            24000          0
TRAINEE                       USD            22000          0
```

READ ステートメントのその他の例

次の例のプログラムを参照してください。

- *READX03 - READ ステートメント*

FIND ステートメント

次のトピックについて説明します。

- FIND ステートメントの使用
- FIND ステートメントの基本構文
- 処理するレコード数の制限
- WHERE 節
- WHERE 節が含まれる FIND ステートメントの例
- IF NO RECORDS FOUND 条件

- FIND ステートメントのその他の例

FIND ステートメントの使用

FIND ステートメントは、指定した検索条件に一致するレコードをデータベースから選択するために使用します。

FIND ステートメントの基本構文

FIND ステートメントの基本構文は次のとおりです。

```
FIND RECORDS IN view WITH field = value
```

または、以下のように短くすることができます。

```
FIND view WITH field = value
```

各項目の意味を次に示します。

view	DEFINE DATA ステートメントで定義されるビューの名前（「*DEFINE DATA ビュー*」を参照）。
field	そのビューで定義されるデータベースフィールドの名前。

*field*に指定できるのは、基礎となる **DDM** で「ディスクリプタ」として定義されているフィールドのみです。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、またはフォネティックディスクリプタの場合もあります。

完全な構文については、FIND ステートメントのドキュメントを参照してください。

処理するレコード数の制限

上述した READ ステートメントの場合と同様に、キーワード FIND の後にカッコで囲んだ数字を指定することによって、処理するレコード数を制限できます。

```
FIND (6) RECORDS IN MYVIEW WITH NAME = 'CLEGG'
```

上記の例では、検索条件に一致する最初の 6 つのレコードだけが処理されます。

リミット表記がない場合は、検索条件に一致するすべてのレコードが処理されます。

注意: FIND ステートメントに WHERE 節（下記参照）が含まれている場合、WHERE 節の結果として拒否されるレコードは制限に対してカウントされません。

WHERE 節

FIND ステートメントの WHERE 節を使用すると、WITH 節で選択したレコードが読み込まれた後、このレコードの処理が実行される*前*に評価される追加の選択条件を指定できます。

WHERE 節が含まれる FIND ステートメントの例

```
** Example 'FINDX01': FIND (with WHERE)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 CITY
END-DEFINE
*
FIND MYVIEW WITH CITY = 'PARIS'
            WHERE JOB-TITLE = 'INGENIEUR COMMERCIAL'
  DISPLAY NOTITLE CITY JOB-TITLE PERSONNEL-ID NAME
END-FIND
END
```

注意: この例では、WITH 節と WHERE 節の*両方*の条件に一致するレコードだけが DISPLAY ステートメントで処理されます。

プログラム FINDX01 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
```

IF NO RECORDS FOUND 条件

WITH 節と WHERE 節に指定した検索条件に一致するレコードが見つからない場合、FIND 処理ループ内のステートメントは実行されません。上記の例では、DISPLAY ステートメントが実行されないため、従業員データは表示されません。

ただし、FIND ステートメントでは IF NO RECORDS FOUND 節も提供されます。これにより、検索条件に一致するレコードがない場合に実行する処理を指定できます。

例：

```
** Example 'FINDX02': FIND (with IF NO RECORDS FOUND)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
END-DEFINE
*
FIND MYVIEW WITH NAME = 'BLACKSMITH'
  IF NO RECORDS FOUND
    WRITE 'NO PERSON FOUND.'
  END-NOREC
  DISPLAY NAME FIRST-NAME
END-FIND
END
```

上記のプログラムでは、NAME フィールドに BLACKSMITH の値があるすべてのレコードが選択されます。 選択された各レコードについて、姓（NAME）と名前（FIRST-NAME）が表示されます。 NAME = 'BLACKSMITH' のレコードがファイルに見つからない場合は、IF NO RECORDS FOUND 節内の WRITE ステートメントが実行されます。

プログラム FINDX02 の出力：

```
Page      1                                                 04-11-11  14:15:54

        NAME              FIRST-NAME
-------------------- --------------------

NO PERSON FOUND.
```

FIND ステートメントのその他の例

次の例のプログラムを参照してください。

- *FINDX07 - FIND（複数節を含む）*
- *FINDX08 - FIND（LIMIT を含む）*
- *FINDX09 - FIND（*NUMBER、*COUNTER、*ISN を使用）*
- *FINDX10 - FIND（READ との組み合わせ）*
- *FINDX11 - FIND NUMBER（*NUMBER を含む）*

HISTOGRAM ステートメント

次のトピックについて説明します。

HISTOGRAM ステートメントの使用

HISTOGRAM ステートメントは、1 つのデータベースフィールドの値だけを読み込むか、または指定した検索条件に一致するレコード数を決定するために使用します。

HISTOGRAM ステートメントでは、HISTOGRAM ステートメントに指定されたもの以外のデータベースフィールドへのアクセスは提供されません。

HISTOGRAM ステートメントの基本構文

HISTOGRAM ステートメントの基本構文は次のとおりです。

```
HISTOGRAM VALUE IN view FOR field
```

または、以下のように短くすることができます。

```
HISTOGRAM view FOR field
```

各項目の意味を次に示します。

view	DEFINE DATA ステートメントで定義されるビューの名前（「*DEFINE DATA ビュー*」を参照）。
field	そのビューで定義されるデータベースフィールドの名前。

完全な構文については、HISTOGRAM ステートメントのドキュメントを参照してください。

読み込む値の数の制限

READ ステートメントの場合と同様に、キーワード HISTOGRAM の後にカッコで囲んだ数字を指定することによって、読み込まれるレコード数を制限できます。

```
HISTOGRAM (6) MYVIEW FOR NAME
```

上記の例では、フィールド NAME の最初の 6 つの値だけが読み込まれます。

リミット表記がない場合は、すべての値が読み込まれます。

STARTING／ENDING 節

READ ステートメントと同様に、HISTOGRAM ステートメントでも、開始値と終了値を指定して読み込む値の範囲を絞り込むために、STARTING FROM 節と ENDING AT（またはTHRU）節が提供されます。

例：

```
HISTOGRAM MYVIEW FOR NAME STARTING from 'BOUCHARD'
HISTOGRAM MYVIEW FOR NAME STARTING from 'BOUCHARD' ENDING AT 'LANIER'
HISTOGRAM MYVIEW FOR NAME from 'BLOOM' THRU 'ROESER'
```

WHERE 節

HISTOGRAM ステートメントでも、値が読み込まれた後、その値の処理が実行される*前*に評価される追加の選択条件を指定できる WHERE 節が提供されます。 WHERE 節に指定するフィールドは、HISTOGRAM ステートメントの主節のフィールドと同じである必要があります。

HISTOGRAM ステートメントの例

```
** Example 'HISTOX01': HISTOGRAM
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 CITY
END-DEFINE
*
LIMIT 8
HISTOGRAM MYVIEW CITY STARTING FROM 'M'
  DISPLAY NOTITLE CITY 'NUMBER OF/PERSONS' *NUMBER *COUNTER
END-HISTOGRAM
END
```

上記のプログラムでは、システム変数 *NUMBER と *COUNTER も HISTOGRAM ステートメントで評価され、DISPLAY ステートメントで出力されます。 *NUMBER には最後に読み込まれた値が含まれるデータベースレコードの数が入り、*COUNTER には読み込まれた値の合計数が入ります。

プログラム HISTOX01 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
```

MULTI-FETCH 節

このセクションは、Adabas データベースのマルチフェッチレコード検索機能について説明します。

このセクションで説明するマルチフェッチ機能は、Adabas に対してのみサポートされます。DB2 データベースのマルチフェッチレコード検索機能の詳細については、『*データベース管理システムインターフェイス*』ドキュメントの「*Natural for DB2*」セクションの「*複数行の処理*」を参照してください。

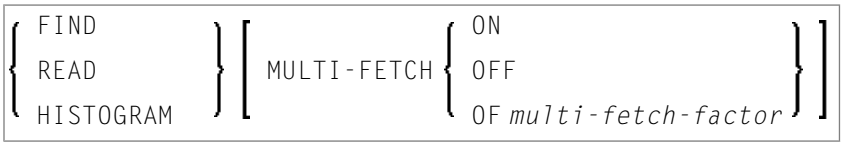
次のトピックについて説明します。

- マルチフェッチ機能の目的
- マルチフェッチ使用時の考慮事項
- マルチフェッチバッファのサイズ
- マルチフェッチの TEST DBLOG サポート

マルチフェッチ機能の目的

標準モードの Natural は、単一のデータベースコールでは複数のレコードを読み込みまず、フェッチごとに 1 つのレコードを取得するモードで常に稼働します。 このような稼働は堅実で安定していますが、大量のデータベースレコードの処理には時間がかかる場合があります。

これらのプログラムのパフォーマンスを向上させるために、MULTI-FETCH 節を `FIND`、`READ`、または `HISTOGRAM` の各ステートメントで使用できます。 これにより、データベースアクセス 1 回当たりに読み込まれるレコードの数を指定する数値であるマルチフェッチ要因を定義できます。



multi-fetch-factor はフォーマット整数（I4）の定数または変数のいずれかです。

ステートメント実行時には、1 よりも大きい *multi-fetch-factor* がデータベースステートメントに与えられているかどうかがランタイムによってチェックされます。

multi-fetch-factor の値に応じて、動作は次のようになります。

負の値	ランタイムエラーが発生します。
0 または 1	データベースコールは、アクセスごとに 1 つのレコードを取得する通常のモードで継続します。
2 以上	データベースコールは、補助バッファへの単一のデータベースアクセスで 10 個など複数のレコードを読み込むためにダイナミックに行われます（マルチフェッチバッファ）。 正常に読み込まれると、最初のレコードが、基礎となるデータビューに転送されます。 次のループの実行時には、データベースにアクセスしなくても、マルチフェッチバッファからデータビューにレコードが直接読み込まれます。 すべてのデータがマルチフェッチバッファからフェッチされた後は、次のループで次のレコードセットのデータベースからの読み込みが行われます。エンドオブレコード、ESCAPE、STOP などによってデータベースループが終了した場合、マルチフェッチバッファの内容は解放されます。

マルチフェッチ使用時の考慮事項

- マルチフェッチアクセスはブラウズループ、つまりレコードが 「ホールドなし」 で読み込まれる場合のためにのみサポートされます。
- プログラムは、各ループごとにデータベースから「新しい」レコードを取得するのではなく、最も新しいマルチフェッチアクセスで取得したイメージで動作します。
- READ/HISTOGRAM ステートメントに対してループの再位置決めが発生すると、その時点のマルチフェッチバッファの内容が解放されます。
- READ/HISTOGRAM ステートメントに対してダイナミックな方向変更（IN DYNAMIC...SEQUENCE）がコーディングされている場合は、マルチフェッチ機能を利用できず、コンパイル時に該当する構文エラーが発生します。
- FIND ループの最初のレコードは、最初の S1 コマンドで検索されます。 Adabas のマルチフェッチはあらゆる種類の Lx コマンドに対して定義されているため、2 つ目のレコードから使用することができます。
- データベースループに占有されるマルチフェッチバッファの大きさは、次のルールに従って決定されます。

  ((レコードバッファ長 + ISN バッファエントリ長) * マルチフェッチ要因) + 4 + ヘッダー長
  =
  ((ビューフィールドのサイズ + 20) * マルチフェッチ要因) + 4 + 128

  必要なスペースを小さくするため、次の条件下ではランタイムでマルチフェッチ要因が自動的に減らされます。

  - READ（2）.. などの 「ループ制限」 の方が小さい場合。ただし、WHERE 節が含まれない場合のみ。
  - 「ISN 数」 の方が小さい場合。FIND ステートメントの場合のみ。
  - 処理後のレコードバッファまたは ISN バッファのサイズが 32KB よりも大きくなる場合。

さらに、次の条件下では、ランタイムにマルチフェッチオプションが完全に無視されます。

- マルチフェッチ要因に 1 以下の値が入っている場合。
- マルチフェッチバッファが利用不可であるか、または十分なフリースペースがない場合。詳細については、次の「*マルチフェッチバッファのサイズ*」を参照してください。

マルチフェッチバッファのサイズ

マルチフェッチの目的で使用できるストレージの量を制御するために、マルチフェッチバッファの最大サイズを制限することができます。

NATPARM 定義内で、パラメータマクロ NTDS を使用して、スタティックな割り当てを次のように作成できます。

```
NTDS MULFETCH,nn
```

セッション開始時には、次のようにプロファイルパラメータ DS を使うこともできます。

```
DS=(MULFETCH,nn)
```

nn は、マルチフェッチの目的に割り当てることが可能な全体サイズを KB 単位で表します。 値は 0～1024 の範囲で設定でき、デフォルト値は 64 です。 大きな値を設定しても、必ずしもそのサイズのバッファが割り当てられるわけではありません。これは、マルチフェッチハンドラによって、マルチフェッチデータベースステートメントの実行に何が実際に必要であるかに応じて、ダイナミックアロケーションとサイズ変更が行われるためです。 Natural セッションでマルチフェッチデータベースステートメントが一度も実行されない場合は、設定された値にかかわらずマルチフェッチバッファは作成されません。

値 0 を指定した場合は、データベースアクセスステートメントに MULTI-FETCH OF .. 節が含まれているかどうかにかかわらず、マルチフェッチ処理は完全に無効になります。 この操作を行うことで、使用可能なストレージを現在の環境で十分に確保できない場合、またはデバッグの目的で、すべてのマルチフェッチアクティビティを完全にオフにすることができます。

注意: Adabas の既存の制限により、レコードバッファ または ISN バッファを 32 KB よりも大きくすることはできません。 したがって、FIND、READ、または HISTOGRAM の単一ループでマルチフェッチバッファに必要なスペースは、最大でも 64 KB になります。 マルチフェッチバッファに必要な値の設定は、マルチフェッチで処理するデータベースループのネスト数に応じて異なります。

マルチフェッチの TEST DBLOG サポート

マルチフェッチ関連データベースコールの TEST DBLOG によるサポートの詳細については、『ユーティリティ』ドキュメントの「*DBLOG ユーティリティ*」、「*マルチフェッチを使用する Adabas コマンドの表示*」を参照してください。

データベース処理ループ

このセクションでは、FIND、READ、または HISTOGRAM の各ステートメントの結果としてデータベースから選択されたデータの処理に必要な処理ループについて説明します。

次のトピックについて説明します。

- データベース処理ループの作成
- 処理ループの階層
- 同じファイルにアクセスする FIND ループのネストの例
- READ および FIND ステートメントのネストのその他の例

データベース処理ループの作成

Natural では、FIND、READ、または HISTOGRAM ステートメントの結果としてデータベースから選択されたデータの処理に必要な処理ループが自動的に作成されます。

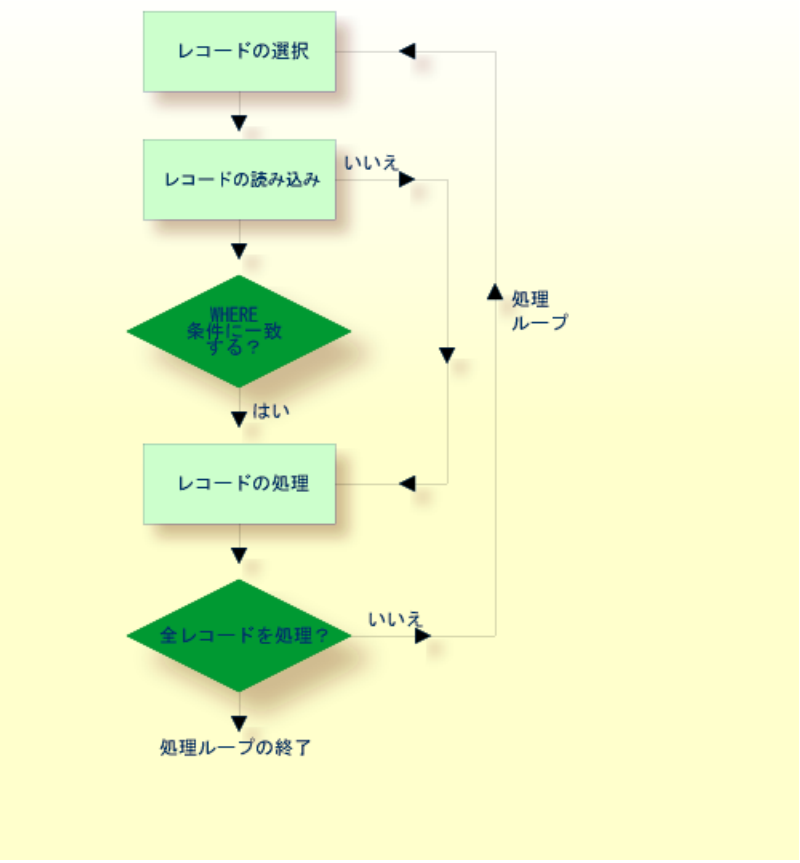
例：

次の例では、FIND ループを使用して、NAME フィールドに ADKINSON という値が含まれるすべてのレコードを EMPLOYEES ファイルから選択し、選択したレコードを処理します。 この例では、選択された各レコードの特定のフィールドを表示する処理が含まれます。

```
** Example 'FINDX03': FIND
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
END-DEFINE
*
FIND MYVIEW WITH NAME = 'ADKINSON'
  DISPLAY NAME FIRST-NAME CITY
END-FIND
END
```

WITH 節に加えて、FIND ステートメントに WHERE 節が含まれる場合は、WITH 節の結果として選択されたレコードのうち、さらに、WHERE 条件に一致するものだけが処理されます。

次の図は、データベース処理ループのフローロジックを示しています。



処理ループの階層

FIND および（または）READ ステートメントを複数使用することによって、次の例に示すように、処理ループの階層が作成されます。

処理ループの階層の例

```
** Example 'FINDX04': FIND  (two FIND statements nested)
************************************************************************
DEFINE DATA LOCAL
1 PERSONVIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 AUTOVIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
  2 MODEL
END-DEFINE
*
EMP. FIND PERSONVIEW WITH NAME = 'ADKINSON'
```

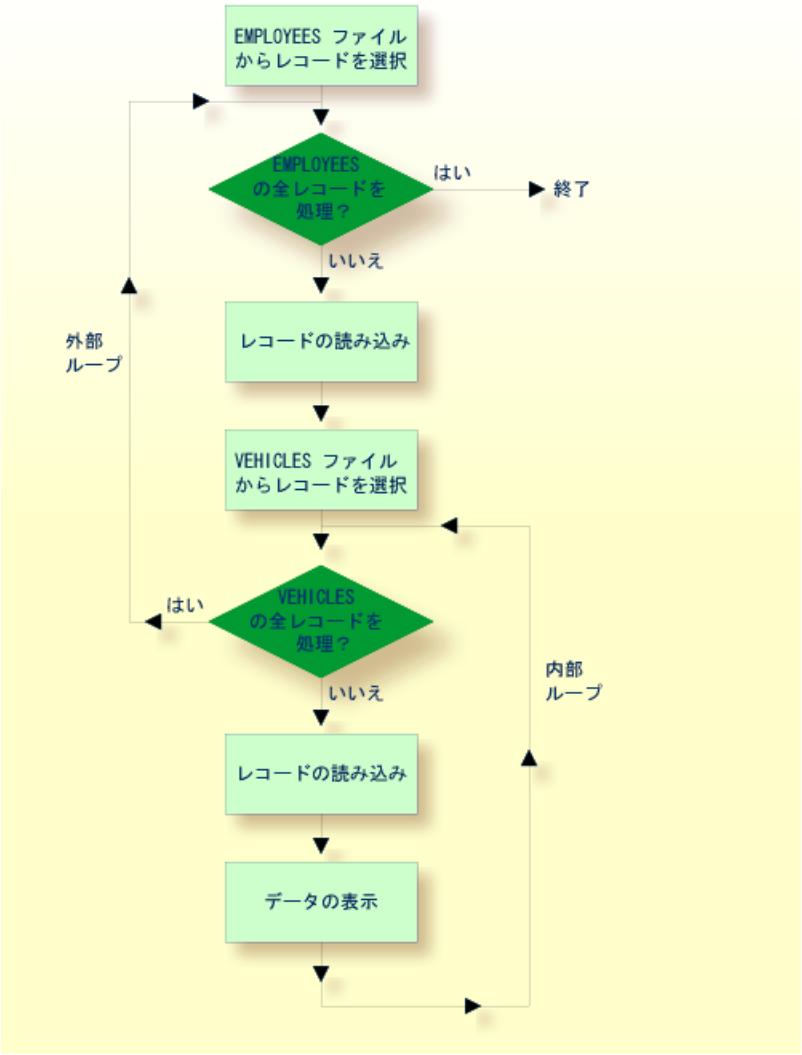
```
  VEH. FIND AUTOVIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMP.)
    DISPLAY NAME MAKE MODEL
  END-FIND
END-FIND
END
```

上記のプログラムでは、ADKINSON という名前のすべての従業員が EMPLOYEES ファイルから選択されます。 選択された各レコード（従業員）は、その後、次のように処理されます。

1. VEHICLES ファイルから車を選択するために、1 つ目の FIND ステートメントで EMPLOYEES ファイルから選択されたレコードの PERSONNEL-ID を選択条件に使用して、2 つ目の FIND ステートメントが実行されます。
2. 選択された各従業員の NAME が表示されます。この情報は EMPLOYEES ファイルから取得されます。 その従業員が所有する各車の MAKE と MODEL も表示されます。この情報は VEHICLES ファイルから取得されます。

2 つ目の FIND ステートメントでは、次の図に示すように、1 つ目の FIND ステートメントの外部処理ループ内に内部処理ループが作成されます。

この図は、上述のプログラム例における処理ループの階層のフローロジックを示しています。



同じファイルにアクセスする FIND ループのネストの例

階層の両方のレベルで同じファイルが使用される処理ループの階層を構成することもできます。

```
** Example 'FINDX05': FIND (two FIND statements on same file nested)
************************************************************************
DEFINE DATA LOCAL
1 PERSONVIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
1 #NAME (A40)
END-DEFINE
*
WRITE TITLE LEFT JUSTIFIED
```

```
  'PEOPLE IN SAME CITY AS:' #NAME / 'CITY:' CITY SKIP 1
*
FIND PERSONVIEW WITH NAME = 'JONES'
                WHERE FIRST-NAME = 'LAUREL'
  COMPRESS NAME FIRST-NAME INTO #NAME
  /*
  FIND PERSONVIEW WITH CITY = CITY
    DISPLAY NAME FIRST-NAME CITY
  END-FIND
END-FIND
END
```

上記のプログラムでは、まず姓が JONES で名前が LAUREL の従業員が EMPLOYEES ファイルからすべて選択されます。 次に、同じ都市に住んでいるすべての従業員が EMPLOYEES ファイルから選択され、そのリストが作成されます。 DISPLAY ステートメントで表示されるすべてのフィールド値は、2 つ目の FIND ステートメントから取得されます。

プログラム FINDX05 の出力：

```

PEOPLE IN SAME CITY AS: JONES LAUREL
CITY: BALTIMORE

        NAME              FIRST-NAME              CITY
-------------------- -------------------- --------------------

JENSON               MARTHA               BALTIMORE
LAWLER               EDDIE                BALTIMORE
FORREST              CLARA                BALTIMORE
ALEXANDER            GIL                  BALTIMORE
NEEDHAM              SUNNY                BALTIMORE
ZINN                 CARLOS               BALTIMORE
JONES                LAUREL               BALTIMORE
```

READ および FIND ステートメントのネストのその他の例

次の例のプログラムを参照してください。

- *READX04 - READ ステートメント（FIND およびシステム変数 *NUMBER と *COUNTER との組み合わせ）*
- *LIMITX01 - LIMIT ステートメント（READ、FIND ループ処理）*

データベース更新 - トランザクション処理

このセクションでは、トランザクションに基づいてNaturalでデータベース更新処理が実行される方法について説明します。

次のトピックについて説明します。

論理トランザクション

Naturalでは、トランザクションに基づいてデータベース更新処理が実行されます。つまり、すべてのデータベース更新要求は論理トランザクション単位で処理されます。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が論理的に一貫性を持っていることを確実にするために、完全に実行されなければならない最小の作業単位です。作業単位の定義はユーザーが行います。

論理トランザクションは、1つ以上のデータベースファイルに関連する1つ以上の更新ステートメント（DELETE、STORE、UPDATE）で構成することができます。 また、論理トランザクションは、複数の Natural プログラムにまたがることもできます。

論理トランザクションは、レコードが「ホールド」状態におかれたときに開始します。Naturalでは、レコードが更新のために読み込まれるとき、例えば、FINDループにUPDATEステートメントやDELETEステートメントが含まれる場合に、この処理が自動的に行われます。

論理トランザクションの終了は、プログラムのEND TRANSACTIONステートメントによって決まります。 このステートメントは、トランザクション内のすべての更新が正常に適用されたことを保証し、トランザクション中に 「ホールド」 状態におかれていたすべてのレコードを解放します。

例：

```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
FIND MYVIEW WITH NAME = 'SMITH'
   DELETE
   END TRANSACTION
END-FIND
END
```

選択された各レコードは 「ホールド」 状態に置かれ、削除され、その後は END TRANSACTION ステートメントが実行されるときに 「ホールド」 状態から解放されます。

注意: Natural 管理者が設定する Natural プロファイルパラメータ ETEOP は、各 Natural プログラムの終了時に Natural で END TRANSACTION ステートメントを生成するかどうかを決定します。 詳細については、Natural 管理者に確認してください。

STORE ステートメントの例

次のプログラム例では、EMPLOYEES ファイルに新しいレコードを追加します。

```
** Example 'STOREX01': STORE  (Add new records to EMPLOYEES file)
*
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOYEE-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID(A8)
  2 NAME        (A20)
  2 FIRST-NAME  (A20)
  2 MIDDLE-I    (A1)
  2 SALARY      (P9/2)
  2 MAR-STAT    (A1)
  2 BIRTH       (D)
  2 CITY        (A20)
  2 COUNTRY     (A3)
*
1 #PERSONNEL-ID (A8)
1 #NAME         (A20)
1 #FIRST-NAME   (A20)
1 #INITIAL      (A1)
1 #MAR-STAT     (A1)
1 #SALARY       (N9)
1 #BIRTH        (A8)
1 #CITY         (A20)
1 #COUNTRY      (A3)
1 #CONF         (A1)   INIT <'Y'>
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
/*********************************************************************
FIP2. FIND NUMBER EMPLOYEE-VIEW WITH PERSONNEL-ID =  #PERSONNEL-ID
/*
IF *NUMBER (FIP2.) > 0
  REINPUT 'PERSON WITH SAME PERSONNEL-ID ALREADY EXISTS'
           MARK 1 AND SOUND ALARM
END-IF
/*********************************************************************
/*  get further information
/*********************************************************************
INPUT
  'ENTER EMPLOYEE DATA'                            ////
  'PERSONNEL-ID             :' #PERSONNEL-ID (AD=IO) /
  'NAME                     :' #NAME         (AD=IO) /
  'FIRST-NAME               :' #FIRST-NAME   (AD=IO) ///
  'INITIAL                  :' #INITIAL              /
  'ANNUAL SALARY            :' #SALARY               /
  'MARITAL STATUS           :' #MAR-STAT             /
  'DATE OF BIRTH (YYYYMMDD) :' #BIRTH                /
  'CITY                     :' #CITY                 /
  'COUNTRY (3 CHARS)        :' #COUNTRY              //
  'ADD THIS RECORD (Y/N)    :' #CONF         (AD=M)
/*********************************************************************
/*  ENSURE REQUIRED FIELDS CONTAIN VALID DATA
/*********************************************************************
IF #SALARY < 10000
  REINPUT TEXT 'ENTER A PROPER ANNUAL SALARY' MARK 2
END-IF
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 3
END-IF
IF NOT(#BIRTH = MASK(YYYYMMDD) AND #BIRTH = MASK(1582-2699))
  REINPUT TEXT 'ENTER CORRECT DATE' MARK 4
END-IF
IF #CITY  = ' '
  REINPUT TEXT 'ENTER A CITY NAME' MARK 5
END-IF
IF #COUNTRY = ' '
  REINPUT TEXT 'ENTER A COUNTRY CODE' MARK 6
END-IF
```

```
  IF NOT (#CONF = 'N' OR= 'Y')
    REINPUT TEXT 'ENTER Y (YES) OR N (NO)' MARK 7
  END-IF
  IF #CONF = 'N'
    ESCAPE TOP
  END-IF
  /*********************************************************************
  /*  add the record with STORE
  /*********************************************************************
  MOVE #PERSONNEL-ID TO EMPLOYEE-VIEW.PERSONNEL-ID
  MOVE #NAME         TO EMPLOYEE-VIEW.NAME
  MOVE #FIRST-NAME   TO EMPLOYEE-VIEW.FIRST-NAME
  MOVE #INITIAL      TO EMPLOYEE-VIEW.MIDDLE-I
  MOVE #SALARY       TO EMPLOYEE-VIEW.SALARY (1)
  MOVE #MAR-STAT     TO EMPLOYEE-VIEW.MAR-STAT
  MOVE EDITED #BIRTH TO EMPLOYEE-VIEW.BIRTH (EM=YYYYMMDD)
  MOVE #CITY         TO EMPLOYEE-VIEW.CITY
  MOVE #COUNTRY      TO EMPLOYEE-VIEW.COUNTRY
  /*
  STP3. STORE RECORD IN FILE EMPLOYEE-VIEW
  /*
  /*********************************************************************
  /*  mark end of logical transaction
  /*********************************************************************
  END OF TRANSACTION
  RESET INITIAL #CONF
END-REPEAT
END
```

プログラム STOREX01 の出力：

```

ENTER A PERSONNEL ID AND NAME (OR 'END' TO END)

PERSONNEL ID :

NAME         :
FIRST NAME   :
```

レコードホールドロジック

Natural を Adabas で使用する場合、更新するレコードは、END TRANSACTION または BACKOUT TRANSACTION ステートメントが発行されるまで、あるいはトランザクションタイムリミットを超えるまで、「ホールド」 状態におかれます。

1人のユーザーに対してレコードが「ホールド」状態におかれると、他のユーザーはそのレコードを更新できません。 同じレコードを更新しようとする他のユーザーは、最初のユーザーがそのトランザクションを終了またはバックアウトしてレコードが 「ホールド」 から解放されるまで 「待ち」 状態におかれます。

ユーザーが待ち状態におかれることを防ぐために、セッションパラメータ WH（ホールドレコードの待機）を使用できます。『パラメータリファレンス』を参照してください。

プログラムで更新ロジックを使用するときには、以下の点を考慮する必要があります。

- レコードをホールド状態における最大時間は、Adabas のトランザクションタイムリミット（Adabas パラメータ TT）によって決まります。このタイムリミットを超えると、エラーメッセージが表示され、最後の END TRANSACTION 以降に行われたすべてのデータベース更新が取り消されます。
- ホールドされるレコード件数およびトランザクションタイムリミットは、トランザクションのサイズ、つまり、プログラムにおける END TRANSACTION ステートメントの配置に影響されます。 どこで END TRANSACTION を発行するかを決めるときには、再スタート機能を考慮する必要があります。例えば、処理中のレコードの大多数が更新*されない*場合は、レコードの「ホールド」状態の制御には GET ステートメントが効率的です。これにより、複数の END TRANSACTION ステートメントの発行が回避され、ホールドされる ISN の数が少なくなります。 大きなファイルを処理するときは、GET ステートメントでは追加の Adabas コールが必要となることに注意してください。 GET ステートメントの例を次に示します。
- レコードを「ホールド」状態に置く処理は、Natural 管理者が設定するプロファイルパラメータ RI（リリース ISN）によって制御されます。

ホールドロジックの例

```
** Example 'GETX01': GET (put single record in hold with UPDATE stmt)
**
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 SALARY (1)
END-DEFINE
*
RD. READ EMPLOY-VIEW BY NAME
  DISPLAY EMPLOY-VIEW
  IF SALARY (1) > 1500000
    /*
    GE. GET EMPLOY-VIEW *ISN (RD.)
    /*
    WRITE '=' (50) 'RECORD IN HOLD:' *ISN(RD.)
    COMPUTE SALARY (1) = SALARY (1) * 1.15
    UPDATE (GE.)
    END TRANSACTION
  END-IF
END-READ
END
```

トランザクションのバックアウト

アクティブな論理トランザクション中、つまり、END TRANSACTION ステートメントが発行される前に、BACKOUT TRANSACTION ステートメントを使用してトランザクションをキャンセルすることができます。 このステートメントを実行すると、これまでに適用されたすべての更新（追加または削除されたすべてのレコードを含む）が削除され、トランザクションによってホールドされていたすべてのレコードが解放されます。

トランザクションの再スタート

END TRANSACTION 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。 トランザクション処理が異常終了する場合は、GET TRANSACTION DATA ステートメントでこの情報を読み取り、トランザクションを再スタートするときにどこで処理を再開するかを確認できます。

トランザクションデータを使用したトランザクション再スタートの例

次のプログラムは、EMPLOYEES ファイルと VEHICLES ファイルを更新します。 再スタート処理の後で、正常に処理された最後の EMPLOYEES レコードがユーザーに通知されます。 ユーザーは、その EMPLOYEES レコードから処理を再開できます。 再スタート処理の前に正常に更新された最後の VEHICLES レコードを、トランザクションの再スタートメッセージに含めるように設定することもできます。

```
** Example 'GETTRX01': GET TRANSACTION
*
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
01 PERSON VIEW OF EMPLOYEES
  02 PERSONNEL-ID      (A8)
  02 NAME              (A20)
  02 FIRST-NAME        (A20)
  02 MIDDLE-I          (A1)
  02 CITY              (A20)
01 AUTO VIEW OF VEHICLES
  02 PERSONNEL-ID      (A8)
  02 MAKE              (A20)
  02 MODEL             (A20)
*
01 ET-DATA
  02 #APPL-ID          (A8) INIT <' '>
  02 #USER-ID          (A8)
  02 #PROGRAM          (A8)
  02 #DATE             (A10)
  02 #TIME             (A8)
  02 #PERSONNEL-NUMBER (A8)
END-DEFINE
```

```
*
GET TRANSACTION DATA #APPL-ID #USER-ID #PROGRAM
                     #DATE    #TIME    #PERSONNEL-NUMBER
*
IF  #APPL-ID NOT = 'NORMAL'     /* if last execution ended abnormally
AND #APPL-ID NOT = ' '
  INPUT (AD=OIL)
    //  20T '*** LAST SUCCESSFUL TRANSACTION ***' (I)
    /   20T '***********************************'
    /// 25T      'APPLICATION:' #APPL-ID
    /   32T             'USER:' #USER-ID
    /   29T          'PROGRAM:' #PROGRAM
    /   24T     'COMPLETED ON:' #DATE 'AT' #TIME
    /   20T 'PERSONNEL NUMBER:' #PERSONNEL-NUMBER
END-IF
*
REPEAT
  /*
  INPUT (AD=MIL) // 20T 'ENTER PERSONNEL NUMBER:' #PERSONNEL-NUMBER
  /*
  IF #PERSONNEL-NUMBER = '99999999'
    ESCAPE BOTTOM
  END-IF
  /*
  FIND1. FIND PERSON WITH PERSONNEL-ID = #PERSONNEL-NUMBER
    IF NO RECORDS FOUND
      REINPUT 'SPECIFIED NUMBER DOES NOT EXIST; ENTER ANOTHER ONE.'
    END-NOREC
    FIND2. FIND AUTO WITH PERSONNEL-ID = #PERSONNEL-NUMBER
      IF NO RECORDS FOUND
        WRITE 'PERSON DOES NOT OWN ANY CARS'
        ESCAPE BOTTOM
      END-NOREC
      IF *COUNTER (FIND2.) = 1     /* first pass through the loop
        INPUT (AD=M)
          /   20T 'EMPLOYEES/AUTOMOBILE DETAILS' (I)
          /   20T '----------------------------'
          /// 20T 'NUMBER:' PERSONNEL-ID (AD=O)
          /   22T   'NAME:' NAME ' ' FIRST-NAME ' ' MIDDLE-I
          /   22T   'CITY:' CITY
          /   22T   'MAKE:' MAKE
          /   21T  'MODEL:' MODEL
        UPDATE (FIND1.)            /* update the EMPLOYEES file
      ELSE                         /* subsequent passes through the loop
        INPUT NO ERASE (AD=M IP=OFF) //////// 28T MAKE / 28T MODEL
      END-IF
      /*
      UPDATE (FIND2.)              /* update the VEHICLES file
      /*
      MOVE *APPLIC-ID TO #APPL-ID
      MOVE *INIT-USER TO #USER-ID
      MOVE *PROGRAM   TO #PROGRAM
```

```
      MOVE *DAT4E     TO #DATE
      MOVE *TIME      TO #TIME
      /*
      END TRANSACTION #APPL-ID #USER-ID #PROGRAM
                      #DATE    #TIME    #PERSONNEL-NUMBER
      /*
    END-FIND                  /* for VEHICLES    (FIND2.)
  END-FIND                    /* for EMPLOYEES   (FIND1.)
END-REPEAT                    /* for REPEAT
*
STOP                          /* Simulate abnormal transaction end
END TRANSACTION 'NORMAL  '
END
```

ACCEPT/REJECT を使用したレコードの選択

このセクションでは、ユーザー指定の論理条件に基づいてレコードを選択するために使用する ACCEPT ステートメントおよび REJECT ステートメントについて説明します。

次のトピックについて説明します。

- ACCEPT および REJECT とともに使用可能なステートメント
- ACCEPT ステートメントの例
- ACCEPT/REJECT ステートメントの論理条件基準
- AND 演算子を指定した ACCEPT ステートメントの例
- OR 演算子を指定した REJECT ステートメントの例
- ACCEPT/REJECT ステートメントのその他の例

ACCEPT および REJECT とともに使用可能なステートメント

ACCEPT ステートメントおよび REJECT ステートメントは、次のデータベースアクセスステートメントと連携して使用できます。

- READ
- FIND
- HISTOGRAM

ACCEPT ステートメントの例

```
** Example 'ACCEPX01': ACCEPT IF
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 CURR-CODE (1:1)
  2 SALARY    (1:1)
END-DEFINE
*
READ (20) MYVIEW BY NAME WHERE CURR-CODE (1) = 'USD'
ACCEPT IF SALARY (1) >= 40000
  DISPLAY NAME JOB-TITLE SALARY (1)
END-READ
END
```

プログラム ACCEPX01 の出力：

```
 Page      1                                                 04-11-11  11:11:11

        NAME                  CURRENT            ANNUAL
                             POSITION            SALARY
  ------------------- ------------------------- ----------

  ADKINSON            DBA                          46700
  ADKINSON            MANAGER                      47000
  ADKINSON            MANAGER                      47000
  AFANASSIEV          DBA                          42800
  ALEXANDER           DIRECTOR                     48000
  ANDERSON            MANAGER                      50000
  ATHERTON            ANALYST                      43000
  ATHERTON            MANAGER                      40000
```

ACCEPT/REJECT ステートメントの論理条件基準

ACCEPT ステートメントおよび REJECT ステートメントを使用すると、READ ステートメントの WITH 節および WHERE 節で指定した論理条件に加えて、論理条件を指定できます。

ACCEPT/REJECT ステートメントの IF 節の論理条件基準は、レコードが選択されて読み込まれた*後*で評価されます。

論理条件演算子には次のものがあります。詳細については、「*論理条件基準*」を参照してください。

EQUAL	EQ	:=
NOT EQUAL TO	NE	¬=
LESS THAN	LT	<
LESS EQUAL	LE	<=
GREATER THAN	GT	>
GREATER EQUAL	GE	>=

ACCEPT/REJECT ステートメントの論理条件基準は、ブール演算子 AND、OR、および NOT で結合することもできます。 さらに、論理グループを示すためにカッコを使用することもできます。次の例を参照してください。

AND 演算子を指定した ACCEPT ステートメントの例

次のプログラムは、ACCEPT ステートメントでのブール演算子 AND の使用を示しています。

```
** Example 'ACCEPX02': ACCEPT IF ... AND ...
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 CURR-CODE (1:1)
  2 SALARY    (1:1)
END-DEFINE
*
READ (20) MYVIEW BY NAME WHERE CURR-CODE (1) = 'USD'
ACCEPT IF  SALARY (1) >= 40000
         AND SALARY (1) <= 45000
  DISPLAY NAME JOB-TITLE SALARY (1)
END-READ
END
```

プログラム ACCEPX02 の出力：

```
Page      1                                                  04-12-14  12:22:01

        NAME                  CURRENT            ANNUAL
                             POSITION            SALARY
-------------------- ------------------------- ----------

AFANASSIEV           DBA                            42800
ATHERTON             ANALYST                        43000
ATHERTON             MANAGER                        40000
```

OR 演算子を指定した REJECT ステートメントの例

次のプログラムは、REJECT ステートメントでブール演算子 OR を使用したものです。論理演算子が逆になっているため、前の例の ACCEPT ステートメントと同じ出力が生成されます。

```
** Example 'ACCEPX03': REJECT IF ... OR ...
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 CURR-CODE (1:1)
  2 SALARY    (1:1)
END-DEFINE
*
READ (20) MYVIEW BY NAME WHERE CURR-CODE (1) = 'USD'
REJECT IF SALARY (1) < 40000
         OR SALARY (1) > 45000
  DISPLAY NAME JOB-TITLE SALARY (1)
END-READ
END
```

プログラム ACCEPX03 の出力：

```
Page      1                                                  04-12-14  12:26:27

        NAME                  CURRENT            ANNUAL
                             POSITION            SALARY
-------------------- ------------------------- ----------

AFANASSIEV           DBA                            42800
ATHERTON             ANALYST                        43000
ATHERTON             MANAGER                        40000
```

ACCEPT/REJECT ステートメントのその他の例

次の例のプログラムを参照してください。

- ***ACCEPX04 - ACCEPT IF ... LESS THAN ...***
- ***ACCEPX05 - ACCEPT IF ... AND ...***
- ***ACCEPX06 - REJECT IF ... OR ...***

AT START/END OF DATA ステートメント

このセクションでは、AT START OF DATA ステートメントおよび AT END OF DATA ステートメントの使用について説明します。

次のトピックについて説明します。

AT START OF DATA ステートメント

AT START OF DATA ステートメントは、データベース処理ループで一連のレコードの最初のレコードが読み込まれた後に実行する処理を指定するために使用します。

AT START OF DATA ステートメントは、処理ループ内に指定する必要があります。

AT START OF DATA 処理で出力が生成される場合は、*最初のフィールド値の前*に出力されます。デフォルトでは、この出力は左揃えでページに表示されます。

AT END OF DATA ステートメント

AT END OF DATA ステートメントは、データベース処理ループのすべてのレコードが処理された後に実行する処理を指定するために使用します。

AT END OF DATA ステートメントは、処理ループ内に指定する必要があります。

AT END OF DATA 処理で出力が生成される場合は、*最後のフィールド値の後*に出力されます。 デフォルトでは、この出力は左揃えでページに表示されます。

AT START OF DATA/AT END OF DATA ステートメントの例

次のプログラム例は、AT START OF DATA ステートメントおよび AT END OF DATA ステートメントの使用を示しています。

時刻を表示するために、Natural のシステム変数 *TIME が AT START OF DATA ステートメントに組み込まれています。

最後に選択された従業員の名前を表示するために、Natural のシステム関数 OLD が AT END OF DATA ステートメントに組み込まれています。

```
** Example 'ATSTAX01': AT START OF DATA
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 JOB-TITLE
  2 INCOME  (1:1)
    3 CURR-CODE
    3 SALARY
    3 BONUS (1:1)
END-DEFINE
*
WRITE TITLE 'XYZ EMPLOYEE ANNUAL SALARY AND BONUS REPORT' /
READ (3) MYVIEW BY CITY STARTING FROM 'E'
  DISPLAY GIVE SYSTEM FUNCTIONS
          NAME (AL=15) JOB-TITLE (AL=15) INCOME (1)
  /*
 AT START OF DATA
    WRITE 'RUN TIME:' *TIME /
  END-START
  AT END OF DATA
    WRITE / 'LAST PERSON SELECTED:' OLD (NAME) /
  END-ENDDATA
END-READ
*
AT END OF PAGE
  WRITE / 'AVERAGE SALARY:' AVER (SALARY(1))
END-ENDPAGE
END
```

プログラムが次の出力を生成します。

```
                  XYZ EMPLOYEE ANNUAL SALARY AND BONUS REPORT

     NAME           CURRENT                 INCOME
                    POSITION
                                  CURRENCY   ANNUAL     BONUS
                                    CODE     SALARY
--------------- --------------- -------- ---------- ----------

RUN TIME: 12:43:19.1

DUYVERMAN       PROGRAMMER      USD           34000          0
PRATT           SALES PERSON    USD           38000       9000
MARKUSH         TRAINEE         USD           22000          0

LAST PERSON SELECTED: MARKUSH
```

```
AVERAGE SALARY:      31333
```

AT START OF DATA/AT END OF DATA のその他の例

次の例のプログラムを参照してください。

- *ATENDX01 - AT END OF DATA*
- *ATSTAX02 - AT START OF DATA*
- *WRITEX09 - WRITE（AT END OF DATA との組み合わせ）*

Unicode データ

Natural を使用すると、Adabas データベースのワイド文字フィールド（フォーマット W）にアクセスできます。

次のトピックについて説明します。

- データ定義モジュール
- アクセスコンフィグレーション
- 制限事項

データ定義モジュール

Adabas のワイド文字フィールド（W）は、Natural フォーマット U（Unicode）にマッピングされます。

フォーマット U の Natural フィールドの長さの定義は、フォーマット W の Adabas フィールドのサイズの半分に一致します。例えば、長さ 200 の Adabas のワイド文字フィールドは、Natural の (U100) にマッピングされます。

アクセスコンフィグレーション

Natural は Adabas からデータを受け取り、共通のエンコードとして UTF-16 を使用してデータを Adabas に送ります。

このエンコードは OPRB パラメータで指定され、オープン要求で Adabas に送られます。これはワイド文字フィールドに使用され、Adabas ユーザーセッション全体を通して適用されます。

制限事項

照合ディスクリプタはサポートされていません。

Adabas および Unicode のサポートの詳細については、該当する Adabas 製品のドキュメントを参照してください。

27 SQL データベースのデータへのアクセス

原則として、『*Adabas データベースのデータへのアクセス*』ドキュメントに記載されている機能とその例は、Natural がサポートする SQL データベースにも該当します。

相違点がある場合は、個別のデータベースアクセスステートメントに関するドキュメントの「*データベース固有の考慮事項*」（『*ステートメント*』ドキュメントを参照）または各 Natural パラメータに関するドキュメント（『*パラメータリファレンス*』を参照）に詳細が記載されています。

また、Natural for DB2 では、DB2 および SQL/DS データベース管理システムにアクセスするための特別な一連のステートメントが提供されます。 詳細については、『*データベース管理システムインターフェイス*』ドキュメントの次のアドオン製品に関する説明を参照してください。

- *Natural for DB2*
- *Natural for SQL/DS*
- *SQL 版 Natural Gateway*

28 VSAM データベースのデータへのアクセス

原則として、「*Adabas データベースのデータへのアクセス*」ドキュメントに記載されている機能と例は、VSAM データベースにも該当します。

相違点がある場合は、個別のデータベースアクセスステートメントに関するドキュメントの「*データベース固有の考慮事項*」（『ステートメント』ドキュメントを参照）または各 Natural パラメータに関するドキュメント（『パラメータリファレンス』を参照）に詳細が記載されています。

詳細については、『データベース管理システムインターフェイス』ドキュメントの次のアドオン製品に関する説明を参照してください。

- *Natural for VSAM*

29 DL/I データベースのデータへのアクセス

原則として、「*Adabas データベースのデータへのアクセス*」ドキュメントに記載されている機能と例は、DL/I データベースにも該当します。

相違点がある場合は、個別のデータベースアクセスステートメントに関するドキュメントの「*データベース固有の考慮事項*」（『ステートメント』ドキュメントを参照）または各 Natural パラメータに関するドキュメント（『パラメータリファレンス』を参照）に詳細が記載されています。

詳細については、『データベース管理システムインターフェイス』ドキュメントの次のアドオン製品に関する説明を参照してください。

- *Natural for DL/I*

30 データ出力制御

This part describes how to proceed if a Natural program is to produce multiple reports. さらに、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、つまり、データの表示方法について、さまざまな面から説明します。

- レポート指定 - (*rep*) 表記
- 出力ページのレイアウト
- **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
- ページタイトル、改ページ、空行
- 列ヘッダー
- フィールドの出力に影響を与えるパラメータ
- 編集マスク - **EM** パラメータ
- 垂直表示

31 レポート指定 - (rep) 表記

(*rep*) は、ステートメントを適用できる出力レポートの識別子です。

レポート指定の使用

Natural プログラムで複数のレポートを作成する場合、最初のレポート（レポート 0）以外のレポートを出力する出力ステートメント（下記の「*ステートメントに関する考慮事項*」を参照）ごとに、表記 (*rep*) を指定する必要があります。

0～31 の値を指定できます。

この表記は、バッチモードで作成するレポート、Com-plete、CMS、IMS/TM、TIAM 環境のレポート、または CICS、TSO、UTM 環境で Natural Advanced Facilities を使用している場合のレポートにのみ適用されます。

(*rep*) の値には、DEFINE PRINTER ステートメントを使用して割り当てた論理名も指定できます。下記の*例 2* を参照してください。

ステートメントに関する考慮事項

表記 (*rep*) は以下の出力ステートメントで使用できます。

AT END OF PAGE | AT TOP OF PAGE | DISPLAY | EJECT | FORMAT | NEWPAGE | PRINT | SKIP | SUSPEND IDENTICAL SUPPRESS | WRITE | WRITE TITLE | WRITE TRAILER

レポート指定の例

例 **1 -** 複数のレポート

```
DISPLAY (1) NAME ...
WRITE (4) NAME ...
```

例 **2 -** 論理名の使用

```
DEFINE PRINTER (LIST=5) OUTPUT 'LPT1'
WRITE (LIST) NAME ...
```

32 出力ページのレイアウト

次のトピックについて説明します。

レポートレイアウトに影響するステートメント

レポートのレイアウトに影響するステートメントは、以下のとおりです。

ステートメント	機能
WRITE TITLE	ページタイトル、つまりページの先頭に出力するテキストを指定できます。 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。
WRITE TRAILER	ページトレーラ、つまりページの下に出力するテキストを指定できます。 デフォルトでは、トレーラ行は中央揃えで下線なしです。
AT TOP OF PAGE	レポートの新しいページが開始されるときに常に実行される処理を指定できます。 この処理による出力は、ページタイトルの下に出力されます。
AT END OF PAGE	ページ終了条件が発生したときに常に実行される処理を指定できます。 この処理による出力は、`WRITE TRAILER` ステートメントで指定したページトレーラの下に出力されます。
AT START OF DATA	データベース処理ループで最初のレコードが読み込まれた後に実行される処理を指定します。 この処理による出力は、最初のフィールド値の前に出力されます。
AT END OF DATA	処理ループのすべてのレコードが処理された後に実行される処理を指定します。 この処理からの出力は、最後のフィールド値の直後に出力されます。
DISPLAY / WRITE	これらのステートメントで、読み込まれたフィールド値を出力する際のフォーマットを制御します。 「*DISPLAY および WRITE ステートメント*」を参照してください。

データ出力に対する `AT START OF DATA` および `AT END OF DATA` ステートメントに関連する説明は、データベースアクセスの「*AT START/END OF DATA ステートメント*」を参照してください。 上記リストの他のステートメントについては、「*データ出力制御*」の各セクションを参照してください。

一般的なレイアウトの例

以下のプログラム例は、出力ページの一般的なレイアウトを示しています。

```
** Example 'OUTPUX01': Several sections of output
************************************************************************
DEFINE DATA LOCAL
1 EMP-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
END-DEFINE
*
WRITE TITLE   '********** Page Title **********'
WRITE TRAILER '********** Page Trailer **********'
*
AT TOP OF PAGE
  WRITE '===== Top of Page ====='
END-TOPPAGE
AT END OF PAGE
  WRITE '===== End of Page ====='
END-ENDPAGE
*
READ (10) EMP-VIEW BY NAME
  /*
  DISPLAY NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD)
  /*
  AT START OF DATA
    WRITE '>>>>> Start of Data >>>>>'
  END-START
  AT END OF DATA
    WRITE '<<<<< End of Data <<<<<'
  END-ENDDATA
END-READ
END
```

プログラム OUTPUX01 の出力：

```
                    ********** Page Title **********
===== Top of Page =====
        NAME              FIRST-NAME         DATE
                                              OF
                                            BIRTH
-------------------- -------------------- ----------

>>>>> Start of Data >>>>>
```

```
ABELLAN              KEPA                 1961-04-08
ACHIESON             ROBERT               1963-12-24
ADAM                 SIMONE               1952-01-30
ADKINSON             JEFF                 1951-06-15
ADKINSON             PHYLLIS              1956-09-17
ADKINSON             HAZEL                1954-03-19
ADKINSON             DAVID                1946-10-12
ADKINSON             CHARLIE              1950-03-02
ADKINSON             MARTHA               1970-01-01
ADKINSON             TIMMIE               1970-03-03
<<<<< End of Data <<<<<
                  ********** Page Trailer **********
===== End of Page =====
```

33 DISPLAY および WRITE ステートメント

次のトピックについて説明します。

DISPLAY ステートメント

DISPLAY ステートメントでは列形式の出力が作成されます。つまり、1 つのフィールドの値が 1 つの列の下に縦に出力されます。 複数のフィールドが出力される場合、つまり複数の列が作成される場合、これらの列は互いに隣り合って横に出力されます。

表示されるフィールドの順番は、DISPLAY ステートメントに指定したフィールド名の順番で決定されます。

以下のプログラムの DISPLAY ステートメントでは、従業員ごとに、最初に従業員番号、次に名前、その次に職種が表示されます。

```
** Example 'DISPLX01': DISPLAY
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH
  2 JOB-TITLE
END-DEFINE
*
READ (3) VIEWEMP BY BIRTH
  DISPLAY PERSONNEL-ID NAME JOB-TITLE
END-READ
END
```

プログラム DISPLX01 の出力：

```
Page      1                                                  04-11-11  14:15:54

PERSONNEL         NAME                  CURRENT
   ID                                   POSITION
--------- -------------------- -------------------------

30020013  GARRET               TYPIST
30016112  TAILOR               WAREHOUSEMAN
20017600  PIETSCH              SECRETARY
```

出力レポートに表示される列の順番を変更するには、DISPLAY ステートメントのフィールド名を単に並べ換えます。 例えば、従業員名、職種、従業員番号の順番でリストする場合、適切な DISPLAY ステートメントは、以下のようになります。

```
** Example 'DISPLX02': DISPLAY
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH
  2 JOB-TITLE
END-DEFINE
*
READ (3) VIEWEMP BY BIRTH
  DISPLAY NAME JOB-TITLE PERSONNEL-ID
END-READ
END
```

プログラム DISPLX02 の出力：

```
Page      1                                                 04-11-11  14:15:54

        NAME                  CURRENT           PERSONNEL
                             POSITION              ID
-------------------- ------------------------- ---------

GARRET               TYPIST                    30020013
TAILOR               WAREHOUSEMAN              30016112
PIETSCH              SECRETARY                 20017600
```

ヘッダーが各列の上に出力されます。このヘッダーに影響するさまざまな方法については、『*列ヘッダー*』ドキュメントを参照してください。

WRITE ステートメント

WRITE ステートメントは、フリーフォーマット（列なし）の出力を作成するために使用します。DISPLAY ステートメントとは対照的に、WRITE ステートメントには以下が適用されます。

- 必要に応じて、自動的に行送りを行います。つまり、現在の出力行に納まらないフィールドまたはテキスト要素は、自動的に次の行に出力されます。
- ヘッダーは作成されません。
- マルチプルバリューフィールドの値は、互いに隣り合って横に出力されます。縦には出力されません。

以下の2つのプログラム例は、DISPLAY ステートメントと WRITE ステートメントの基本的な違いを示しています。

『*垂直表示*』ドキュメントの「***DISPLAY と WRITE の組み合わせ***」で説明しているように、2つのステートメントを組み合わせて使用することもできます。

DISPLAY ステートメントの例

```
** Example 'DISPLX03': DISPLAY
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 SALARY (1:3)
END-DEFINE
*
READ (2) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY NAME FIRST-NAME SALARY (1:3)
END-READ
END
```

プログラム DISPLX03 の出力：

```
Page      1                                              04-11-11  14:15:54

        NAME              FIRST-NAME        ANNUAL
                                            SALARY
-------------------- -------------------- ----------

JONES                VIRGINIA                 46000
                                              42300
                                              39300
JONES                MARSHA                   50000
                                              46000
                                              42700
```

WRITE ステートメントの例

```
** Example 'WRITEX01': WRITE
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 SALARY (1:3)
END-DEFINE
*
READ (2) VIEWEMP BY NAME STARTING FROM 'JONES'
  WRITE NAME FIRST-NAME SALARY (1:3)
END-READ
END
```

プログラム WRITEX01 の出力：

```
Page      1                                          04-11-11  14:15:55

JONES                VIRGINIA                  46000      42300      39300
JONES                MARSHA                    50000      46000      42700
```

列の間隔 - SF パラメータと nX 表記

デフォルトでは、DISPLAY ステートメントで出力される列は互いに *1* 文字の空白で区切られます。

セッションパラメータ SF を使用すると、DISPLAY ステートメントで出力される列の間に挿入するデフォルトの空白文字数を指定できます。 空白文字数は 1～30 の任意の値に設定できます。

パラメータは、FORMAT ステートメント を使用してレポート全体に適用するか、または DISPLAY ステートメントを使用して要素レベルではなくステートメントレベルで指定することができます。

DISPLAY ステートメントの *n*X 表記で、2 つの列の間に挿入する空白文字数（*n*）を指定できます。 *n*X 表記は、SF パラメータによる指定を上書きします。

```
** Example 'DISPLX04': DISPLAY (with nX)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH
  2 JOB-TITLE
END-DEFINE
*
FORMAT SF=3
READ (3) VIEWEMP BY BIRTH
  DISPLAY PERSONNEL-ID NAME 5X JOB-TITLE
END-READ
END
```

プログラム DISPLX04 の出力：

上記のプログラム例では、以下の出力が作成されます。最初の 2 つの列は FORMAT ステートメントの SF パラメータにより 3 文字の空白で区切られています。2 番目と 3 番目の列は、DISPLAY ステートメントの 5X 表記により 5 文字の空白で区切られています。

```
Page      1                                                 04-11-11  14:15:54

PERSONNEL           NAME                      CURRENT
   ID                                         POSITION
---------    --------------------     -------------------------

30020013    GARRET                   TYPIST
30016112    TAILOR                   WAREHOUSEMAN
20017600    PIETSCH                  SECRETARY
```

*n*X 表記は WRITE ステートメントでも使用可能であり、個々の出力要素間に空白を挿入します。

```
WRITE PERSONNEL-ID 5X NAME 3X JOB-TITLE
```

上記のステートメントでは、フィールド PERSONNEL-ID と NAME の間に 5 文字の空白が挿入され、NAME と JOB-TITLE の間には 3 文字の空白が挿入されます。

タブ設定 - nT 表記

*n*T 表記は DISPLAY および WRITE ステートメントで使用可能であり、この表記を使用して出力要素を出力する位置を指定できます。

```
** Example 'DISPLX05': DISPLAY (with nT)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
END-DEFINE
*
READ (3) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY 5T NAME 30T FIRST-NAME
END-READ
END
```

プログラム DISPLX05 の出力：

上記のプログラムでは、以下の出力が作成されます。フィールド NAME はページの左マージンから数えて 5 桁目から始まり、フィールド FIRST-NAME は 30 桁目から始まります。

```
Page      1                                              04-11-11  14:15:54

            NAME                 FIRST-NAME
    --------------------     --------------------

    JONES                    VIRGINIA
    JONES                    MARSHA
    JONES                    ROBERT
```

行送り - スラッシュ表記

DISPLAY または WRITE ステートメントでスラッシュ（/）を使用すると、行送りを行うことができます。

- DISPLAY ステートメントでは、スラッシュにより*フィールド間*および*テキスト内*で行送りが行われます。
- WRITE ステートメントでは、スラッシュが*フィールド間*に指定された場合にのみ行送りが行われます。テキスト内に指定された場合は、通常のテキスト文字と同様に処理されます。

フィールド間に指定する場合は、スラッシュの両側に空白が必要です。

行送りを複数回行う場合は、複数のスラッシュを指定します。

例**1 - DISPLAY** ステートメントの行送り：

```
** Example 'DISPLX06': DISPLAY (with slash '/')
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 DEPARTMENT
END-DEFINE
*
READ (3) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY NAME / FIRST-NAME 'DEPART-/MENT' DEPARTMENT
END-READ
END
```

プログラム DISPLX06 の出力：

上記の DISPLAY ステートメントでは、フィールド NAME の各値の後、およびテキスト DEPART-MENT 内で行送りが行われます。

```
Page      1                                                 04-11-11  14:15:54

        NAME          DEPART-
     FIRST-NAME        MENT
-------------------- -------

JONES                SALE
VIRGINIA
JONES                MGMT
MARSHA
JONES                TECH
ROBERT
```

例**2 - WRITE** ステートメントの行送り：

```
** Example 'WRITEX02': WRITE (with line advance)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 DEPARTMENT
```

```
END-DEFINE
*
READ (3) VIEWEMP BY NAME STARTING FROM 'JONES'
  WRITE NAME / FIRST-NAME 'DEPART-/MENT' DEPARTMENT //
END-READ
END
```

プログラム WRITEX02 の出力：

上記の WRITE ステートメントでは、フィールド NAME の各値の後に 1 行、フィールド DEPARTMENT の各値の後に 2 行行送りが行われますが、テキスト DEPART-/MENT 内では行送りは行われません。

```
Page      1                                                  04-11-11  14:15:55

JONES
VIRGINIA             DEPART-/MENT SALE


JONES
MARSHA               DEPART-/MENT MGMT


JONES
ROBERT               DEPART-/MENT TECH
```

例**3 - DISPLAY** および **WRITE** の行送り：

```
** Example 'DISPLX21': DISPLAY (usage of slash '/' in DISPLAY and WRITE)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 ADDRESS-LINE (1)
END-DEFINE
*
WRITE TITLE LEFT JUSTIFIED UNDERLINED
       *TIME
   5X  'PEOPLE LIVING IN SALT LAKE CITY'
   21X 'PAGE:' *PAGE-NUMBER /
   15X 'AS OF' *DAT4E //
*
WRITE TRAILER UNDERLINED 'REGISTER OF' / 'SALT LAKE CITY'
*
READ (2) EMPLOY-VIEW WITH CITY = 'SALT LAKE CITY'
  DISPLAY  NAME /
           FIRST-NAME
```

```
           'HOME/CITY' CITY
           'STREET/OR BOX NO.' ADDRESS-LINE (1)
  SKIP 1
END-READ
END
```

プログラム DISPLX21 の出力：

```
14:15:54.6     PEOPLE LIVING IN SALT LAKE CITY                      PAGE:      1
               AS OF 11/11/2004


------------------------------------------------------------------------------
        NAME                 HOME                STREET
     FIRST-NAME              CITY              OR BOX NO.
-------------------- -------------------- --------------------

ANDERSON             SALT LAKE CITY       3701 S. GEORGE MASON
JENNY

SAMUELSON            SALT LAKE CITY       7610 W. 86TH STREET
MARTIN

                                  REGISTER OF
                                SALT LAKE CITY
------------------------------------------------------------------------------
```

DISPLAY および WRITE ステートメントの他の例

次の例のプログラムを参照してください。

- *DISPLX13 - DISPLAY（WRITE を使用している WRITEX08 と比較）*
- *WRITEX08 - WRITE（DISPLAY を使用している DISPLX13 と比較）*
- *DISPLX14 - DISPLAY（AL、SF、および nX を使用）*
- *WRITEX09 - WRITE（AT END OF DATA との組み合わせ）*

34 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

次のトピックについて説明します。

インデックス表記の使用

インデックス表記 (*n*:*n*) を使用すると、出力するマルチプルバリューフィールドの値の個数またはピリオディックグループのオカレンス数を指定できます。

例えば、DDM EMPLOYEES のフィールド INCOME は、ある従業員が会社に雇用されてからの年ごとの年間収入記録を保持するピリオディックグループです。

これらの年間収入は年順に保持されています。最新の年収は、オカレンス 1 に含まれています。

従業員の最近 3 年間の年間収入、つまりオカレンス 1〜3 を表示する場合は、DISPLAY または WRITE ステートメントのフィールド名の後に (1:3) の表記を指定します。以下のプログラム例を参照してください。

DISPLAY ステートメントのインデックス表記の例

```
** Example 'DISPLX07': DISPLAY (with index notation)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH
  2 INCOME  (1:3)
    3 CURR-CODE
    3 SALARY
    3 BONUS (1:1)
END-DEFINE
*
READ (3) VIEWEMP BY BIRTH
  DISPLAY PERSONNEL-ID NAME INCOME (1:3)
  SKIP 1
END-READ
END
```

プログラム DISPLX07 の出力：

DISPLAY ステートメントでは、マルチプルバリューフィールドの複数の値が縦に出力されることに注意してください。

```
Page      1                                                 04-11-11  14:15:54

PERSONNEL         NAME                      INCOME
   ID
                                 CURRENCY   ANNUAL     BONUS
                                   CODE     SALARY
--------- -------------------- -------- ---------- ----------

30020013  GARRET               UKL            4200          0
                               UKL            4150          0
                                                 0          0

30016112  TAILOR               UKL            7450          0
                               UKL            7350          0
                               UKL            6700          0

20017600  PIETSCH              USD           22000          0
                               USD           20200          0
                               USD           18700          0
```

WRITE ステートメントでは複数の値が縦ではなく横に表示されるため、これにより行あふれおよび不要な行送りが発生する場合があります。

ピリオディックグループ全体ではなくピリオディックグループ内の単一フィールド（SALARY など）のみを使用し、さらに以下の例の NAME と JOB-TITLE の間のようにスラッシュ（/）を挿入して行送りを行うと、扱いやすいレポートフォーマットになります。

WRITE ステートメントのインデックス表記の例

```
** Example 'WRITEX03': WRITE (with index notation)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH
  2 JOB-TITLE
  2 SALARY (1:3)
END-DEFINE
*
READ (3) VIEWEMP BY BIRTH
  WRITE PERSONNEL-ID NAME / JOB-TITLE SALARY (1:3)
  SKIP 1
END-READ
END
```

プログラム WRITEX03 の出力：

```
Page      1                                                 04-11-11  14:15:55

30020013 GARRET
TYPIST                          4200       4150          0

30016112 TAILOR
WAREHOUSEMAN                    7450       7350       6700

20017600 PIETSCH
SECRETARY                      22000      20200      18700
```

35 ページタイトル、改ページ、空行

次のトピックについて説明します。

デフォルトのページタイトル

DISPLAY または WRITE ステートメントによるページ出力ごとに、Natural によって自動的に単一のデフォルトタイトル行が生成されます。 このタイトル行には、ページ番号、日付、および時刻が含まれています。

例：

```
WRITE 'HELLO'
END
```

上記のプログラムでは、以下のようにデフォルトのページタイトルを含む出力が作成されます。

```
Page      1                                                 04-12-14  13:19:33

HELLO
```

ページタイトルの省略 - NOTITLE オプション

ページタイトルなしでレポートを出力する場合は、キーワード NOTITLE を DISPLAY または WRITE ステートメントに追加します。

例 **- NOTITLE** を指定した **DISPLAY**：

```
** Example 'DISPLX20': DISPLAY (with NOTITLE)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
END-DEFINE
*
READ (5) EMPLOY-VIEW BY CITY FROM 'BOSTON'
  DISPLAY NOTITLE NAME FIRST-NAME CITY
END-READ
END
```

プログラム DISPLX20 の出力：

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

SHAW                 LESLIE               BOSTON
STANWOOD             VERNON               BOSTON
CREMER               WALT                 BOSTON
PERREAULT            BRENDA               BOSTON
COHEN                JOHN                 BOSTON
```

例 **- NOTITLE** を指定した **WRITE**：

```
WRITE NOTITLE 'HELLO'
END
```

上記のプログラムでは、以下のようにページタイトルを含まない出力が作成されます。

```
HELLO
```

独自ページタイトル定義 - WRITE TITLE ステートメント

Naturalのデフォルトページタイトルの代わりに独自のページタイトルを出力する場合は、`WRITE TITLE` ステートメントを使用します。

以下では次のトピックについて説明します。

- タイトル用テキストの指定
- タイトルの後の空行の指定
- タイトルの桁揃えと下線
- ページ番号付きのタイトル

タイトル用テキストの指定

WRITE TITLE ステートメントで、タイトル用のテキストをアポストロフィで囲んで指定します。

```
WRITE TITLE 'THIS IS MY PAGE TITLE'
WRITE 'HELLO'
END
```

上述したプログラムにより、次の出力が生成されます。

```
                            THIS IS MY PAGE TITLE
HELLO
```

タイトルの後の空行の指定

WRITE TITLE ステートメントの SKIP オプションで、タイトル行のすぐ下に出力される空行の数を指定できます。 キーワード SKIP の後に、挿入する空行の数を指定します。

```
WRITE TITLE 'THIS IS MY PAGE TITLE' SKIP 2
WRITE 'HELLO'
END
```

上述したプログラムにより、次の出力が生成されます。

```
                            THIS IS MY PAGE TITLE



HELLO
```

SKIP は WRITE TITLE ステートメントの一部としてのみではなく、スタンドアロンステートメントとしても使用できます。

タイトルの桁揃えと下線

デフォルトでは、ページタイトルはページの中央に位置付けられ、下線は付きません。

WRITE TITLE ステートメントでは、互いに独立して使用可能な以下のオプションを指定できます。

オプション	効果
LEFT JUSTIFIED	ページタイトルを左揃えで表示します。
UNDERLINED	タイトルを下線付きで表示します。 行サイズの幅に下線が引かれます（Natural プロファイルおよびセッションパラメータ LS も参照）。デフォルトでは、タイトルにハイフン（-）で下線が引かれます。ただし、UC セッションパラメータを使用すると、下線用の文字として使用する他の文字を指定してすることができます（「*タイトルおよびヘッダーの下線用の文字*」を参照）。

以下の例は、LEFT JUSTIFIED および UNDERLINED オプションの効果を示しています。

```
WRITE TITLE LEFT JUSTIFIED UNDERLINED 'THIS IS MY PAGE TITLE'
SKIP 2
WRITE 'HELLO'
END
```

上述したプログラムにより、次の出力が生成されます。

```
THIS IS MY PAGE TITLE
------------------------------------------------------------------------------


HELLO
```

レポートの新しいページが開始されるときには、常に WRITE TITLE ステートメントが実行されます。

ページ番号付きのタイトル

以下の例では、システム変数 *PAGE-NUMBER を WRITE TITLE ステートメントとともに使用して、タイトル行にページ番号を出力します。

```
** Example 'WTITLX01': WRITE TITLE (with *PAGE-NUMBER)
************************************************************************
DEFINE DATA LOCAL
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
  2 YEAR
  2 MAINT-COST (1)
END-DEFINE
*
LIMIT 5
```

```
*
READ VEHIC-VIEW
END-ALL
SORT BY YEAR USING MAKE MAINT-COST (1)
  DISPLAY NOTITLE YEAR MAKE MAINT-COST (1)
  AT BREAK OF YEAR
    MOVE 1 TO *PAGE-NUMBER
    NEWPAGE
  END-BREAK
  /*
  WRITE TITLE LEFT JUSTIFIED
        'YEAR:' YEAR 15X 'PAGE' *PAGE-NUMBER
END-SORT
END
```

プログラム WTITLX01 の出力：

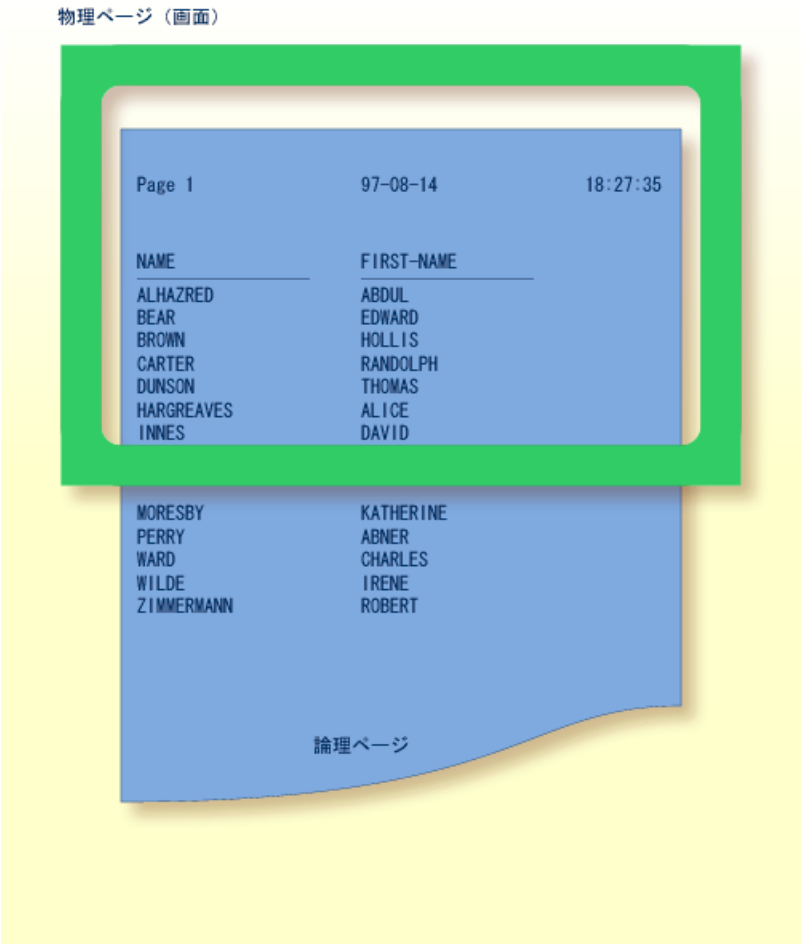
```
YEAR:  1980                PAGE      1
YEAR          MAKE          MAINT-COST
----- -------------------- ----------

 1980 RENAULT                   20000
 1980 RENAULT                   20000
 1980 PEUGEOT                   20000
```

論理ページおよび物理ページ

*論理*ページは、Naturalプログラムによって作成される出力です。*物理*ページは、出力が表示される端末画面です。または、出力がプリントされる用紙 1 枚の場合もあります。

論理ページのサイズは、Natural プログラムによって出力される行数で決定されます。

1画面分より多い行数が出力された場合、論理ページは物理画面を超過し、残りの行は次の画面に表示されます。



注意: 画面の下に表示される情報（WRITE TRAILER または AT END OF PAGE ステートメントで作成される出力など）が次の画面に出力される場合は、下記で説明しているセッションパラメータ PS を使用して、論理ページのサイズを適宜に減らします。

ページサイズ - PS パラメータ

パラメータ PS（Natural レポートのページサイズ）を使用して、レポートの論理ページ当たりの最大行数を指定します。

PS パラメータで指定した行数に達すると、改ページが発生します。ただし、NEWPAGE または EJECT ステートメントで改ページが制御されていない場合に限られます。下記の「*EJ パラメータで制御する改ページ*」を参照してください。

PS パラメータは、システムコマンド GLOBALS を使用してセッションレベルで設定するか、または以下のステートメントを使用してプログラム内で設定できます。

レポートレベル：

- FORMAT PS=*nn*

ステートメントレベル：

- DISPLAY (PS=*nn*)
- WRITE (PS=*nn*)
- WRITE TITLE (PS=*nn*)
- WRITE TRAILER (PS=*nn*)
- INPUT (PS=*nn*)

改ページ

以下のいずれかの方法で改ページできます。

- EJ パラメータで制御する改ページ
- EJECT または NEWPAGE ステートメントで制御する改ページ
- n 行より少ない行数が残っている場合のページ換えまたは新しいページ

これらの方法について以下で説明します。

EJ パラメータで制御する改ページ

セッションパラメータ EJ（ページ換え）を使用して、ページ換えを実行するかどうかを指定します。 デフォルトでは EJ=ON が適用され、指定に従ってページ換えが実行されます。

EJ=OFF を指定すると、改ページ情報は無視されます。 これは、ページ換えを必要としないテスト実行時に用紙を節約するために役立ちます。

EJ パラメータは、次の例のように、システムコマンド GLOBALS を使用してセッションレベルで設定できます。

```
GLOBALS EJ=OFF
```

EJ パラメータの設定は、EJECT ステートメントによって上書きされます。

EJECT または NEWPAGE ステートメントで制御する改ページ

以下では次のトピックについて説明します。

- 次ページにタイトルまたはヘッダーを生成しない改ページ
- ページ終了処理およびページ開始処理を行う改ページ

次ページにタイトルまたはヘッダーを生成しない改ページ

EJECT ステートメントでは、次のページにタイトル行またはヘッダー行を生成しないで改ページが行われます。新しい物理ページは、ページ開始処理またはページ終了処理（WRITE TRAILER、AT END OF PAGE、WRITE TITLE、AT TOP OF PAGE、*PAGE-NUMBER などの処理）を実行しないで、開始されます。

EJECT ステートメントは EJ パラメータの設定を上書きします。

ページ終了処理およびページ開始処理を行う改ページ

NEWPAGE ステートメントでは、関連するページ終了処理およびページ開始処理とともに改ページが行われます。 指定されている場合には、トレーラ行が表示されます。 DISPLAY または WRITE ステートメント で NOTITLE オプションが指定されていない場合には（上記参照）、デフォルトまたはユーザー指定のタイトル行が新しいページに表示されます。

NEWPAGE ステートメントを使用しない場合、改ページは PS パラメータの設定によって自動的に制御されます。上記の「*ページサイズ - PS パラメータ*」を参照してください。

n 行より少ない行数が残っている場合のページ換えまたは新しいページ

NEWPAGE ステートメントおよび EJECT ステートメントの両方で、WHEN LESS THAN *n* LINES LEFT オプションを指定できます。 このオプションでは、行数（*n*）を指定します。 NEWPAGE および EJECT ステートメントは、ステートメント処理時に現在のページの使用可能行数が *n* 行より少ない場合に実行されます。

例 **1**：

```
FORMAT PS=55
...
NEWPAGE WHEN LESS THAN 7 LINES LEFT
...
```

この例では、ページサイズは 55 行に設定されています。

NEWPAGE ステートメントの処理時に現在のページの残りの行数が 6 行以下の場合にのみ、NEWPAGE ステートメントが実行され、改ページが発生します。 残りの行数が 7 行以上の場合、NEWPAGE ステートメントは実行されず、改ページも発生しません。改ページはその後、セッションパラメータ PS（Natural レポートのページサイズ）の設定に従って、55 行目の後に発生します。

例 **2**：

```
** Example 'NEWPAX02': NEWPAGE (in combination with EJECT and
**                      parameter PS)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
FORMAT PS=15
*
READ (9) EMPLOY-VIEW BY CITY STARTING FROM 'BOSTON'
  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) /
    NEWPAGE
  END-START
  AT BREAK OF CITY
    NEWPAGE WHEN LESS 3 LINES LEFT
  END-BREAK
  DISPLAY CITY (IS=ON) NAME JOB-TITLE
END-READ
END
```

タイトル付きの新しいページ

NEWPAGE ステートメントでは、WITH TITLE オプションも使用できます。 このオプションを使用しない場合、デフォルトタイトルが新しいページの先頭に表示されるか、または WRITE TITLE ステートメントや NOTITLE 節が実行されます。

NEWPAGE ステートメントの WITH TITLE オプションにより、独自に選択したタイトルでこれらを上書きすることができます。 WITH TITLE オプションの構文は、WRITE TITLE ステートメントと同じです。

例：

```
NEWPAGE WITH TITLE LEFT JUSTIFIED 'PEOPLE LIVING IN BOSTON:'
```

以下のプログラムは、セッションパラメータPS（Naturalレポートのページサイズ）とNEWPAGEステートメントの使用法を示しています。さらに、システム変数*PAGE-NUMBERを使用して現在のページ番号を表示しています。

```
** Example 'NEWPAX01': NEWPAGE
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 DEPT
END-DEFINE
*
FORMAT PS=20
READ (5) VIEWEMP BY CITY STARTING FROM 'M'
  DISPLAY NAME 'DEPT' DEPT 'LOCATION' CITY
  AT BREAK OF CITY
    NEWPAGE WITH TITLE LEFT JUSTIFIED
           'EMPLOYEES BY CITY - PAGE:' *PAGE-NUMBER
  END-BREAK
END-READ
END
```

プログラムNEWPAX01の出力：

改ページの位置とタイトル行に注意してください。

```
Page      1                                                 04-11-11  14:15:54

        NAME           DEPT        LOCATION
-------------------- ------ --------------------

FICKEN               TECH10 MADISON
KELLOGG              TECH10 MADISON
ALEXANDER            SALE20 MADISON
```

2ページ目：

```
EMPLOYEES BY CITY - PAGE:      2
        NAME           DEPT        LOCATION
-------------------- ------ --------------------

DE JUAN              SALE03 MADRID
DE LA MADRID         PROD01 MADRID
```

3 ページ目：

```
EMPLOYEES BY CITY - PAGE: 3
```

ページトレーラ - WRITE TRAILER ステートメント

以下では次のトピックについて説明します。

- ページトレーラの指定
- 論理ページサイズの考慮事項
- ページトレーラの桁揃えと下線

ページトレーラの指定

WRITE TRAILER ステートメントは、ページの下に（アポストロフィで囲んだ）テキストを出力するために使用されます。

```
WRITE TRAILER 'THIS IS THE END OF THE PAGE'
```

ステートメントは、ページ終了条件が検出されたとき、または SKIP や NEWPAGE ステートメントの結果として実行されます。

論理ページサイズの考慮事項

ページ終了条件は DISPLAY または WRITE ステートメント全体が処理された後でのみチェックされるため、論理ページサイズ（DISPLAY または WRITE ステートメントによって出力される行数）によっては、WRITE TRAILER ステートメントが実行される前に、出力ページの物理サイズを超過する可能性があります。

実際に物理ページの下にページトレーラが表示されるようにするには、PS セッションパラメータを使用して、論理ページサイズを物理ページサイズより少ない値に設定する必要があります。

ページトレーラの桁揃えと下線

デフォルトでは、ページトレーラはページの中央に位置付けられ、下線は付きません。

WRITE TRAILERステートメントでは、互いに独立して使用可能な以下のオプションを指定できます。

オプション	効果
LEFT JUSTIFIED	ページトレーラを左揃えで表示します。
UNDERLINED	行サイズの幅に下線が引かれます（Naturalプロファイルおよびセッションパラメータ LS も参照）。 デフォルトでは、タイトルにハイフン（-）で下線が引かれます。 ただし、UCセッションパラメータを使用すると、下線用の文字として使用する他の文字を指定してすることができます（「タイトルおよびヘッダーの下線用の文字」を参照）。

以下の例は、WRITE TRAILER ステートメントの LEFT JUSTIFIED および UNDERLINED オプションの使用法を示しています。

例 **1**：

```
WRITE TRAILER LEFT JUSTIFIED UNDERLINED 'THIS IS THE END OF THE PAGE'
```

例 **2**：

```
** Example 'WTITLX02': WRITE TITLE AND WRITE TRAILER
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 ADDRESS-LINE (1)
END-DEFINE
*
WRITE TITLE LEFT JUSTIFIED UNDERLINED
       *TIME
   5X  'PEOPLE LIVING IN SALT LAKE CITY'
   21X 'PAGE:' *PAGE-NUMBER /
   15X 'AS OF' *DAT4E //
*
WRITE TRAILER UNDERLINED 'REGISTER OF' / 'SALT LAKE CITY'
*
READ (2) EMPLOY-VIEW WITH CITY = 'SALT LAKE CITY'
  DISPLAY  NAME /
           FIRST-NAME
           'HOME/CITY' CITY
           'STREET/OR BOX NO.' ADDRESS-LINE (1)
```

```
  SKIP 1
END-READ
END
```

空行の生成 - SKIP ステートメント

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。

例 **1 - WRITE** および **DISPLAY** とともに使用する **SKIP**：

```
** Example 'SKIPX01': SKIP (in conjunction with WRITE and DISPLAY)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 ADDRESS-LINE (1)
END-DEFINE
*
WRITE TITLE LEFT JUSTIFIED UNDERLINED
     'PEOPLE LIVING IN SALT LAKE CITY AS OF' *DAT4E 7X
     'PAGE:' *PAGE-NUMBER
SKIP 3
*
READ (2) EMPLOY-VIEW WITH CITY = 'SALT LAKE CITY'
  DISPLAY NAME / FIRST-NAME CITY ADDRESS-LINE (1)
  SKIP 1
END-READ
END
```

例 **2 - DISPLAY VERT** とともに使用する **SKIP**：

```
** Example 'SKIPX02': SKIP (in conjunction with DISPLAY VERT)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
END-DEFINE
*
READ (2) EMPLOY-VIEW WITH JOB-TITLE = 'SECRETARY'
  DISPLAY NOTITLE VERT
          NAME FIRST-NAME / CITY
```

```
  SKIP 3
END-READ
*
NEWPAGE
*
READ (2) EMPLOY-VIEW WITH JOB-TITLE = 'SECRETARY'
  DISPLAY NOTITLE
          NAME FIRST-NAME / CITY
  SKIP 3
END-READ
END
```

AT TOP OF PAGE ステートメント

AT TOP OF PAGE ステートメントは、レポートの新しいページが始まるときに常に実行される処理を指定するために使用します。

AT TOP OF PAGE 処理で何らかの出力が作成される場合、ページタイトルの下に、間にスキップした行をはさんで出力されます。

デフォルトでは、この出力は左揃えでページに表示されます。

例：

```
** Example 'ATTOPX01': AT TOP OF PAGE
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 MAR-STAT
  2 BIRTH
  2 CITY
  2 JOB-TITLE
  2 DEPT
END-DEFINE
*
LIMIT 10
READ EMPLOY-VIEW BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE (AL=10)
          NAME DEPT JOB-TITLE CITY 5X
          MAR-STAT 'DATE OF/BIRTH' BIRTH (EM=YY-MM-DD)
  /*
  AT TOP OF PAGE
    WRITE /   '-BUSINESS INFORMATION-'
          26X '-PRIVATE INFORMATION-'
  END-TOPPAGE
```

```
END-READ
END
```

プログラム ATTOPX01 の出力：

```
-BUSINESS INFORMATION-                          -PRIVATE INFORMATION-
   NAME    DEPARTMENT  CURRENT      CITY         MARITAL   DATE OF
              CODE     POSITION                   STATUS    BIRTH
---------- ---------- ---------- ----------     ---------- --------

CREMER     TECH10     ANALYST    GREENVILLE     S          70-01-01
MARKUSH    SALE00     TRAINEE    LOS ANGELE     D          79-03-14
GEE        TECH05     MANAGER    CHAPEL HIL     M          41-02-04
KUNEY      TECH10     DBA        DETROIT        S          40-02-13
NEEDHAM    TECH10     PROGRAMMER CHATTANOOG     S          55-08-05
JACKSON    TECH10     PROGRAMMER ST LOUIS       D          70-01-01
PIETSCH    MGMT10     SECRETARY  VISTA          M          40-01-09
PAUL       MGMT10     SECRETARY  NORFOLK        S          43-07-07
HERZOG     TECH05     MANAGER    CHATTANOOG     S          52-09-16
DEKKER     TECH10     DBA        MOBILE         W          40-03-03
```

AT END OF PAGE ステートメント

AT END OF PAGE ステートメントは、ページ終了条件が発生したときに常に実行される処理を指定するために使用します。

AT END OF PAGE 処理により何らかの出力が作成される場合、WRITE TRAILER ステートメントで指定したページトレーラの後に出力されます。

デフォルトでは、この出力は左揃えでページに表示されます。

上記で説明したページトレーラに関わる物理および論理ページサイズと DISPLAY または WRITE ステートメントで出力される行数の考慮事項は、AT END OF PAGE 出力にも同様に適用されます。

例：

```
** Example 'ATENPX01': AT END OF PAGE (with system function available
**                      via GIVE SYSTEM FUNCTIONS in DISPLAY)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY (1)
```

```
END-DEFINE
*
READ (10) EMPLOY-VIEW BY PERSONNEL-ID = '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
          NAME JOB-TITLE 'SALARY' SALARY(1)
  /*
  AT END OF PAGE
    WRITE / 24T 'AVERAGE SALARY: ...' AVER(SALARY(1))
  END-ENDPAGE
END-READ
END
```

プログラム ATENPX01 の出力：

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

CREMER               ANALYST                        34000
MARKUSH              TRAINEE                        22000
GEE                  MANAGER                        39500
KUNEY                DBA                            40200
NEEDHAM              PROGRAMMER                     32500
JACKSON              PROGRAMMER                     33000
PIETSCH              SECRETARY                      22000
PAUL                 SECRETARY                      23000
HERZOG               MANAGER                        48500
DEKKER               DBA                            48000


AVERAGE SALARY: ...      34270
```

その他の例

次の例のプログラムを参照してください。

- ***DISPLX21 - DISPLAY*** *（スラッシュ '/' を使用、* ***WRITE*** *と比較）*

36 列ヘッダー

次のトピックについて説明します。

デフォルトの列ヘッダー

デフォルトでは、DISPLAY ステートメントによるデータベースフィールドの各出力は、DDM のフィールドに定義されているデフォルトの列ヘッダー付きで表示されます。

```
** Example 'DISPLX01': DISPLAY
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH
  2 JOB-TITLE
END-DEFINE
*
READ (3) VIEWEMP BY BIRTH
  DISPLAY PERSONNEL-ID NAME JOB-TITLE
END-READ
END
```

プログラム DISPLX01 の出力：

上記のプログラム例では、デフォルトのヘッダーを使用して、以下の出力が作成されます。

```
Page      1                                                 04-11-11  14:15:54

PERSONNEL         NAME                  CURRENT
   ID                                   POSITION
--------- -------------------- -------------------------

30020013  GARRET               TYPIST
30016112  TAILOR               WAREHOUSEMAN
20017600  PIETSCH              SECRETARY
```

デフォルトの列ヘッダーの省略 - NOHDR オプション

列ヘッダーなしでレポートを出力する場合は、DISPLAY ステートメントにキーワード NOHDR を追加します。

```
DISPLAY NOHDR PERSONNEL-ID NAME JOB-TITLE
```

独自の列ヘッダーの定義

デフォルトヘッダーの代わりに独自の列ヘッダーを出力する場合、フィールドの直前に '*text*'（アポストロフィ付き）を指定します。*text* がそのフィールドに使用するヘッダーとなります。

```
** Example 'DISPLX08': DISPLAY  (with column title in 'text')
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH
  2 JOB-TITLE
END-DEFINE
*
READ (3) VIEWEMP BY BIRTH
  DISPLAY PERSONNEL-ID
          'EMPLOYEE' NAME
          'POSITION' JOB-TITLE
END-READ
END
```

プログラム DISPLX08 の出力：

上記のプログラムには、フィールド NAME のヘッダー EMPLOYEE、およびフィールド JOB-TITLE のヘッダー POSITION が含まれています。フィールド PERSONNEL-ID についてはデフォルトヘッダーが使用されます。 プログラムが次の出力を生成します。

```
Page      1                                                  04-11-11  14:15:54

PERSONNEL       EMPLOYEE               POSITION
   ID
--------- -------------------- -------------------------

30020013  GARRET               TYPIST
30016112  TAILOR               WAREHOUSEMAN
20017600  PIETSCH              SECRETARY
```

NOTITLE と NOHDR の組み合わせ

ページタイトルも列ヘッダーもないレポートを作成するには、NOTITLEおよびNOHDRオプションを以下の順番で一緒に指定します。

```
DISPLAY NOTITLE NOHDR PERSONNEL-ID NAME JOB-TITLE
```

列ヘッダーの中央揃え - HC パラメータ

デフォルトでは、列ヘッダーは列の上に中央揃えで出力されます。HCパラメータを使用すると、列ヘッダーの配置を操作することができます。

有効な指定は以下のとおりです。

HC=L	ヘッダーは左揃えで出力されます。
HC=R	ヘッダーは右揃えで出力されます。
HC=C	ヘッダーは中央揃えで出力されます。

HCパラメータをFORMATステートメントで使用してレポート全体に適用させるか、またはDISPLAYステートメントで使用してステートメントレベルおよび要素レベルの両方で適用させることができます。

```
DISPLAY (HC=L) PERSONNEL-ID NAME JOB-TITLE
```

列ヘッダーの幅 - HW パラメータ

HW パラメータを使用して、DISPLAY ステートメントで出力される列の幅を指定します。

有効な指定は以下のとおりです。

HW=ON	DISPLAY 列の幅は、ヘッダーテキストの長さまたはフィールドの長さのどちらか長い方で決定されます。 これは、デフォルトで適用されます。
HW=OFF	DISPLAY 列の幅は、フィールドの長さのみで決定されます。 ただし、HW=OFF はヘッダーを作成しない DISPLAY ステートメント、つまり、NOHDR オプション付きの最初の DISPLAY ステートメントまたは後続の DISPLAY ステートメントにのみ適用されます。

HW パラメータを FORMAT ステートメントで使用してレポート全体に適用させるか、または DISPLAY ステートメントで使用してステートメントレベルおよび要素（フィールド）レベルの両方で適用させることができます。

ヘッダーの充填文字 - FC および GC パラメータ

列の幅をヘッダーではなくフィールド長で決定する場合、FC パラメータを使用して、DISPLAY ステートメントで出力されるヘッダーの両側に列幅全体に渡って表示される*充填文字*を指定します（上記の HW パラメータを参照）。フィールド長で決定しない場合、FC は無視されます。

フィールドグループまたはピリオディックグループを DISPLAY ステートメントで出力する場合、グループ内の個々のフィールドのヘッダーの上に、そのグループに属しているすべてのフィールド列に渡って、*グループヘッダー*が表示されます。 GC パラメータを使用すると、このようなグループヘッダーの両側に表示される*充填文字*を指定できます。

FC パラメータが個々のフィールドのヘッダーに適用されるのに対して、GC パラメータはフィールドグループのヘッダーに適用されます。

パラメータ FC および GC を FORMAT ステートメントで指定してレポート全体に適用させるか、または、DISPLAY ステートメントで指定してステートメントレベルおよび要素（フィールド）レベルの両方で適用させることができます。

```
** Example 'FORMAX01': FORMAT (with parameters FC, GC)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 INCOME (1:1)
    3 CURR-CODE
```

```
    3 SALARY
    3 BONUS (1:1)
END-DEFINE
*
FORMAT FC=* GC=$
*
READ (3) VIEWEMP BY NAME
  DISPLAY NAME (FC==) INCOME (1)
END-READ
END
```

プログラム FORMAX01 の出力：

```
Page      1                                                  04-11-11  14:15:54

========NAME======== $$$$$$$$$$$$INCOME$$$$$$$$$$$$

                     CURRENCY **ANNUAL** **BONUS***
                         CODE     SALARY
-------------------- -------- ---------- ----------

ABELLAN              PTA         1450000          0
ACHIESON             UKL           10500          0
ADAM                 FRA          159980      23000
```

タイトルおよびヘッダーの下線付き文字 - UC パラメータ

デフォルトでは、タイトルおよびヘッダーにハイフン（-）で下線が引かれます。

UC パラメータを使用すると、下線用の文字として使用する別の文字を指定できます。

UC パラメータを FORMAT ステートメントで指定してレポート全体に適用させるか、または DISPLAY ステートメントで指定してステートメントレベルおよび要素（フィールド）レベルの両方で適用させることができます。

```
** Example 'FORMAX02': FORMAT (with parameter UC)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 BIRTH
  2 JOB-TITLE
END-DEFINE
*
FORMAT UC==
```

```
*
WRITE TITLE LEFT JUSTIFIED UNDERLINED 'EMPLOYEES REPORT'
SKIP 1
READ (3) VIEWEMP BY BIRTH
  DISPLAY PERSONNEL-ID (UC=*) NAME JOB-TITLE
END-READ
END
```

上記のプログラムでは、UC パラメータがプログラムレベルおよび要素（フィールド）レベルで指定されています。FORMAT ステートメントで指定された下線文字（=）は、別の下線文字（*）が指定されているフィールド PERSONNEL-ID を除いて、レポート全体に適用されます。

プログラム FORMAX02 の出力：

```
EMPLOYEES REPORT
================================================================================

PERSONNEL         NAME                  CURRENT
   ID                                   POSITION
********* ==================== =========================

30020013  GARRET               TYPIST
30016112  TAILOR               WAREHOUSEMAN
20017600  PIETSCH              SECRETARY
```

列ヘッダーの省略 - スラッシュ表記

アポストロフィスラッシュアポストロフィ（'/'）の表記を使用すると、DISPLAY ステートメントで表示される個々のフィールドのデフォルトの列ヘッダーを省略できます。 NOHDR オプションがすべての列ヘッダーを省略するのに対して、'/' 表記は個々の列ヘッダーを省略するために使用できます。

アポストロフィスラッシュアポストロフィ（'/'）の表記は、DISPLAY ステートメントで、列ヘッダーを省略するフィールドの名前の直前に指定します。

以下の 2 つの例を比較してください。

例 **1**：

```
DISPLAY NAME PERSONNEL-ID JOB-TITLE
```

この場合、3 つすべてのフィールドのデフォルトの列ヘッダーが表示されます。

```
Page      1                                                 04-11-11  14:15:54

        NAME         PERSONNEL          CURRENT
                          ID             POSITION
-------------------- --------- -------------------------

ABELLAN              60008339  MAQUINISTA
ACHIESON             30000231  DATA BASE ADMINISTRATOR
ADAM                 50005800  CHEF DE SERVICE
ADKINSON             20008800  PROGRAMMER
ADKINSON             20009800  DBA
ADKINSON             20011000  SALES PERSON
```

例 **2**：

```
DISPLAY '/' NAME PERSONNEL-ID JOB-TITLE
```

この場合、'/' 表記によってフィールド NAME の列ヘッダーが省略されます。

```
Page      1                                                 04-11-11  14:15:54

                     PERSONNEL          CURRENT
                        ID             POSITION
                     --------- -------------------------

ABELLAN              60008339  MAQUINISTA
ACHIESON             30000231  DATA BASE ADMINISTRATOR
ADAM                 50005800  CHEF DE SERVICE
ADKINSON             20008800  PROGRAMMER
ADKINSON             20009800  DBA
ADKINSON             20011000  SALES PERSON
```

列ヘッダーの他の例

次の例のプログラムを参照してください。

- ***DISPLX15 - DISPLAY（FC、UC を使用）***
- ***DISPLX16 - DISPLAY（'/'、'text'、'text/text' を使用）***

37 フィールドの出力に影響を与えるパラメータ

この章では、フィールドの出力フォーマットの制御に使用できる Natural プロファイルやセッションパラメータの使用法について説明します。

Natural レポートの作成時に使用されるさまざまな標準属性を制御する Natural プロファイルパラメータの概要については、『*オペレーション*』ドキュメントの「*出力レポートおよびワークファイル*」も参照してください。

フィールド出力関連パラメータの概要

Naturalには、フィールドを出力する際のフォーマットを制御するために使用できるプロファイルやセッションパラメータが複数用意されています。

パラメータ	機能
LC、IC、および TC	これらのセッションパラメータで、フィールドの前後またはフィールド値の前に表示する文字を指定できます。
AL および NL	これらのセッションパラメータで、フィールドの出力長を増減できます。
DL	このセッションパラメータで、フォーマットUの英数字マップフィールドのデフォルトの出力長を指定できます。
SG	このセッションパラメータで、負の値をマイナス記号付きまたは記号なしのどちらで表示するかを決定できます。
IS	このセッションパラメータで、後続の同一フィールド値の表示を省略できます。
ZP	このプロファイルおよびセッションパラメータで、0 のフィールド値を表示するかどうかを決定できます。
ES	このセッションパラメータで、DISPLAY または WRITE ステートメントによって生成された空行の表示を省略できます。

これらのパラメータについて以下で説明します。

先頭文字 - LC パラメータ

セッションパラメータ LC では、DISPLAY ステートメントで出力される*フィールドの直前*に表示される先頭文字を指定できます。 出力列の幅は、それに従って拡大します。 1～10 文字を指定できます。

デフォルトでは、値は、英数字フィールドでは左揃え、数値フィールドでは右揃えで表示されます。 これらのデフォルトは AD パラメータで変更できます。『*パラメータリファレンス*』を参照してください。 英数字フィールドに先頭文字を指定すると、文字はフィールド値の直前に表示されます。数値フィールドの場合、先頭文字とフィールド値の間に多数のスペースが表示されることがあります。

LC パラメータは、以下のステートメントで使用できます。

- FORMAT
- DISPLAY

ステートメントレベルと要素レベルで設定できます。

挿入文字 - IC パラメータ

セッションパラメータ IC を使用して、DISPLAY ステートメントで出力される*フィールド値の直前*に挿入され文字を指定します。 1〜10 文字を指定できます。

数値フィールドの場合、挿入文字は、出力される最初の有効桁の直前に挿入されます。指定した文字とフィールド値の間に空白は入りません。 英数字フィールドの場合、IC パラメータの効果は LC パラメータと同じです。

パラメータ LC および IC の両方を 1 つのフィールドに適用することはできません。

IC パラメータは、以下のステートメントで使用できます。

- FORMAT
- DISPLAY

ステートメントレベルと要素レベルで設定できます。

末尾文字 - TC パラメータ

セッションパラメータ TC では、DISPLAY ステートメントで出力される*フィールドのすぐ右側*に表示される末尾文字を指定できます。 出力列の幅は、それに従って拡大します。 1〜10 文字を指定できます。

TC パラメータは、以下のステートメントで使用できます。

- FORMAT
- DISPLAY

ステートメントレベルと要素レベルで設定できます。

出力長 - AL パラメータと NL パラメータ

セッションパラメータ AL では、*英数字フィールドの出力長*を指定できます。NL パラメータでは、*数値フィールドの出力長*を指定できます。 これにより、出力されるフィールドの長さが決定されます。出力されるフィールドの長さは、実際のフィールドの長さより短い場合も長い場合もあります。実際のフィールドの長さは、データベースフィールドの場合はDDM、ユーザー定義変数の場合は DEFINE DATA ステートメントで定義されます。

どちらのパラメータも以下のステートメントで使用できます。

- FORMAT
- DISPLAY
- WRITE
- PRINT
- INPUT

ステートメントレベルと要素レベルで設定できます。

注意: 編集マスクを指定している場合、NL または AL 指定は上書きされます。 編集マスクについては、「*編集マスク - EM パラメータ*」に記載されています。

出力の表示長 - DL パラメータ

注意: DL パラメータの機能を完全に利用するには、Web I/O インターフェイスを使用する必要があります。 端末エミュレーションを使用している場合、例えば、DL で定義した値がフィールド長よりも短いときにフィールド内をスクロールすることはできません。

Unicode 文字列の表示幅は文字列の長さの2倍になる可能性があり、ユーザーは文字列全体を表示できる必要があるため、セッションパラメータ DL で、*フォーマット A または U のフィールドの表示長*を指定できます。 デフォルトはフィールド長であり、例えば、フォーマットおよび長さが U10 の場合、DL が指定されていないときのデフォルトの長さが 10 であるのに対し、表示長は 10～20 になる可能性があります。

このセッションパラメータは以下のステートメントで使用できます。

- FORMAT
- DISPLAY
- WRITE
- PRINT

■ INPUT

ステートメントレベルと要素レベルで設定できます。

セッションパラメータ AL と DL の違いは、AL がフィールドのデータ長を定義するのに対し、DL はフィールドを表示するために使用される画面上の桁数を定義します。 DL セッションパラメータで指定した値がフィールドデータの長さよりも少ない場合、ユーザーは入力フィールド内をスクロールして、フィールドの内容全体を表示することができます。

Web I/O インターフェイスでは、フィールド長よりも少ない長さで DL パラメータを使用することのみをお勧めします。 Natural を端末エミュレーションで実行している場合、フィールド内のスクロールは使用できず、そのため効果は AL パラメータを使用した場合と同じになります。 また、フィールドの内容を変更した場合、表示長を超えるすべての文字が失われます。

注意: DL は A フォーマットフィールドでも同様に使用できます。 これにより、Web I/O インターフェイスとともに使用した場合、編集のコントロールサイズをフィールドの内容よりも小さくすることができます。

例：

```
DEFINE DATA LOCAL
1    #U1 (U10)
1    #U2 (U10)
END-DEFINE
*
#U1 := U'latintxt00'
#U2 := U'特別是伺服器都需要支'
*
INPUT (AD=M) #U1 #U2
END
```

上記のプログラムでは以下の出力が作成されます。フィールド #U2 の内容は不完全です。

```
#U1 latintxt00 #U2 特別是伺服
```

セッションパラメータ DL をフィールド #U2 で使用して適宜に指定すると、このフィールドの内容が正しく表示されます。

```
DEFINE DATA LOCAL
1    #U1 (U10)
1    #U2 (U10)
END-DEFINE
*
#U1 := U'latintxt00'
#U2 := U'特別是伺服器都需要支'
*
```

```
INPUT (AD=M) #U1 #U2 (DL=20)
END
```

結果：

```
#U1 latintxt00 #U2 特別是伺服器都需要支
```

符号の位置 - SG パラメータ

セッションパラメータ SG では、数値フィールドに符号の位置を割り当てるかどうかを決定できます。

- デフォルトでは SG=ON が適用され、数値フィールドに符号の位置が割り当てられます。
- SG=OFF を指定すると、数値フィールドの負の値はマイナス記号（-）なしで出力されます。

SG パラメータは以下のステートメントで使用できます。

- FORMAT
- DISPLAY
- PRINT
- WRITE
- INPUT

ステートメントレベルと要素レベルの両方で設定できます。

注意: 編集マスクを指定している場合、SG 指定は上書きされます。 編集マスクについては、「*編集マスク - EM パラメータ*」に記載されています。

パラメータなしのプログラム例

```
** Example 'FORMAX03': FORMAT (without FORMAT and compare with FORMAX04)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 SALARY (1:1)
  2 BONUS  (1:1,1:1)
END-DEFINE
*
READ (5) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY NAME
          FIRST-NAME
```

```
          SALARY (1:1)
          BONUS  (1:1,1:1)
END-READ
END
```

上記のプログラムにはパラメータ設定が含まれず、出力は以下のようになります。

```
 Page      1                                                 04-11-11  11:11:11

          NAME              FIRST-NAME        ANNUAL     BONUS
                                              SALARY
 -------------------- -------------------- ---------- ----------

 JONES                VIRGINIA                  46000       9000
 JONES                MARSHA                    50000          0
 JONES                ROBERT                    31000          0
 JONES                LILLY                     24000          0
 JONES                EDWARD                    37600          0
```

パラメータ AL、NL、LC、IC、および TC を使用したプログラム例

この例では、セッションパラメータ AL、NL、LC、IC、および TC が使用されています。

```
** Example 'FORMAX04': FORMAT (with parameters AL, NL, LC, TC, IC and
**                      compare with FORMAX03)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 SALARY (1:1)
  2 BONUS  (1:1,1:1)
END-DEFINE
*
FORMAT AL=10 NL=6
*
READ (5) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY NAME            (LC=*)
          FIRST-NAME      (TC=*)
          SALARY (1:1)    (IC=$)
          BONUS (1:1,1:1) (LC=>)
END-READ
END
```

上記のプログラムでは、次の出力が生成されます。個々のパラメータの影響を確認するために、この出力のレイアウトを前のプログラムと比較します。

```
Page      1                                                  04-11-11  11:11:11

   NAME      FIRST-NAME    ANNUAL    BONUS
                           SALARY
----------- ----------- -------- --------

*JONES       VIRGINIA  *   $46000 >   9000
*JONES       MARSHA    *   $50000 >      0
*JONES       ROBERT    *   $31000 >      0
*JONES       LILLY     *   $24000 >      0
*JONES       EDWARD    *   $37600 >      0
```

上記の例でわかるように、AL または NL パラメータで指定した出力長には、LC、IC、および TC パラメータで指定した文字は含まれません。例えば、NAME 列の幅は、フィールド値の 10 文字（AL=10）に先頭文字の 1 文字を足した 11 文字になります。

SALARY および BONUS 列の幅は、フィールド値の 6 文字（NL=6）に、先頭または挿入文字の 1 文字、および符号の位置（SG=ON が適用）の 1 文字を足した 8 文字になります。

重複抑制 - IS パラメータ

セッションパラメータ IS では、WRITE または DISPLAY ステートメントによって作成される連続行で同一情報の表示を抑制できます。

- デフォルトでは、IS=OFF が適用され、同一のフィールド値が表示されます。
- IS=ON を指定した場合、そのフィールドの前の値と同一の値は表示されません。

IS パラメータは以下のステートメントで指定できます。

- レポート全体に適用される FORMAT ステートメント。
- ステートメントレベルおよび要素レベルで適用される DISPLAY または WRITE ステートメント。

ステートメント SUSPEND IDENTICAL SUPPRESS を使用すると、パラメータ IS=ON の効果を 1 件のレコードに対して一時的に無効にできます。詳細については、『ステートメント』ドキュメントを参照してください。

次の 2 つのプログラム例の出力を比較して、IS パラメータの影響を確認します。 2 番目のプログラムでは、NAME フィールドの同一の値は表示されません。

IS パラメータなしのプログラム例

```
** Example 'FORMAX05': FORMAT (without parameter IS
**                      and compare with FORMAX06)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
END-DEFINE
*
READ (3) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY NAME FIRST-NAME
END-READ
END
```

上述したプログラムにより、次の出力が生成されます。

```
  Page      1                                                 04-11-11  11:11:11

           NAME              FIRST-NAME
  ------------------- --------------------

  JONES                VIRGINIA
  JONES                MARSHA
  JONES                ROBERT
```

IS パラメータありのプログラム例

```
** Example 'FORMAX06': FORMAT (with parameter IS
**                      and compare with FORMAX05)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
END-DEFINE
*
FORMAT IS=ON
*
READ (3) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY NAME FIRST-NAME
END-READ
END
```

上述したプログラムにより、次の出力が生成されます。

```
Page      1                                                  04-11-11  11:54:01

        NAME              FIRST-NAME
-------------------- --------------------

JONES                VIRGINIA
                     MARSHA
                     ROBERT
```

ゼロ出力 - ZP パラメータ

プロファイルおよびセッションパラメータ ZP では、ゼロのフィールド値をどのように表示するかを決定します。

- デフォルトでは ZP=ON が適用されるため、0 の各フィールド値に対して 1 つの 0（数値フィールド）またはすべてゼロ（時間フィールド）が表示されます。
- ZP=OFF を指定すると、ゼロの各フィールド値は表示されません。

ZP パラメータは以下のステートメントで指定できます。

- レポート全体に適用される FORMAT ステートメント。
- ステートメントレベルおよび要素レベルで適用される DISPLAY または WRITE ステートメント。

次の 2 つの例のプログラムの出力を比較して、パラメータ ZP および ES の影響を確認します。

空行の省略 - ES パラメータ

セッションパラメータ ES では、DISPLAY または WRITE ステートメントによって作成された空行の出力を省略できます。

- デフォルトでは ES=OFF が適用されるため、すべて空白値を含む行が表示されます。
- ES=ON を指定した場合、すべて空白値を含む DISPLAY または WRITE ステートメントの結果行は表示されません。 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の一部のフィールドを表示するときに、大量の空行が出力される可能性がある場合には、特に有効です。

ES パラメータは以下のステートメントで指定できます。

- レポート全体に適用される FORMAT ステートメント。
- ステートメントレベルで適用される DISPLAY または WRITE ステートメント。

注意: 数値に対して空白の省略を行うには、ES=ON 以外にパラメータ ZP=OFF も関連するフィールドに設定する必要があります。これは、NULL 値を空白に変えてどちらも出力されないようにするためです。

次の 2 つの例のプログラムの出力を比較して、パラメータ ZP および ES の影響を確認します。

パラメータ ZP および ES を使用しないプログラム例

```
** Example 'FORMAX07': FORMAT (without parameter ES and ZP
**                      and compare with FORMAX08)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BONUS (1:2,1:1)
END-DEFINE
*
READ (4) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY NAME FIRST-NAME BONUS (1:2,1:1)
END-READ
END
```

上述したプログラムにより、次の出力が生成されます。

```
Page      1                                              04-11-11  11:58:23

        NAME              FIRST-NAME        BONUS
-------------------- -------------------- ----------

JONES                VIRGINIA                  9000
                                               6750
JONES                MARSHA                       0
                                                  0
JONES                ROBERT                       0
                                                  0
JONES                LILLY                        0
                                                  0
```

パラメータ ZP および ES を使用したプログラム例

```
** Example 'FORMAX08': FORMAT (with parameters ES and ZP
**                      and compare with FORMAX07)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BONUS (1:2,1:1)
END-DEFINE
*
FORMAT ES=ON
*
READ (4) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY NAME FIRST-NAME BONUS (1:2,1:1)(ZP=OFF)
END-READ
END
```

上述したプログラムにより、次の出力が生成されます。

```
Page      1                                                 04-11-11  11:59:09

        NAME              FIRST-NAME        BONUS
-------------------- -------------------- ----------

JONES                VIRGINIA                 9000
                                              6750
JONES                MARSHA
JONES                ROBERT
JONES                LILLY
```

フィールド出力関連パラメータの他の例

パラメータ LC、IC、TC、AL、NL、IS、ZP、ES、および SUSPEND IDENTICAL SUPPRESS ステートメントの他の例については、以下のプログラム例を参照してください。

- ***DISPLX17 - DISPLAY（NL、AL、IC、LC、TC を使用）***
- ***DISPLX18 - DISPLAY（SF、AL、UC、LC、IC、TC のデフォルト設定を使用、DISPLX19 と比較）***
- ***DISPLX19 - DISPLAY（SF、AL、LC、IC、TC を使用、DISPLX18 と比較）***
- ***SUSPEX01 - SUSPEND IDENTICAL SUPPRESS（DISPLAY でパラメータ IS、ES、ZP とともに使用）***

- *SUSPEX02 - SUSPEND IDENTICAL SUPPRESS（DISPLAY でパラメータ IS、ES、ZP とともに使用）。 SUSPEX01 と同一、ただし IS=OFF を使用。*
- *COMPRX03 - COMPRESS*

38 編集マスク - EM パラメータ

この章では、英数字または数値フィールドの編集マスクを指定する方法について説明します。

EM パラメータの使用

セッションパラメータ EM を使用すると、英数字または数値フィールドに対して編集マスクを指定できます。つまり、フィールド値が出力されるフォーマットを文字単位で指定できます。

例：

```
DISPLAY NAME (EM=X^X^X^X^X^X^X^X^X^X)
```

この例で、それぞれの X は表示される英数字フィールド値の 1 文字を表し、それぞれの ^ は空白を表します。 DISPLAY ステートメントで表示された場合、名前 JOHNSON は次のように表示されます。

```
J O H N S O N
```

セッションパラメータ EM は以下のレベルで指定できます。

- レポートレベル（FORMAT ステートメント）。
- ステートメントレベル（DISPLAY、WRITE、INPUT、MOVE EDITED、または PRINT ステートメント）。
- 要素レベル（DISPLAY、WRITE、または INPUT ステートメント）。

セッションパラメータ EM で指定した編集マスクは、**DDM** のフィールドに指定されたデフォルトの編集マスクを上書きします。「*DDM エディタ画面の使用*」の「*拡張フィールド属性の指定*」を参照してください。

EM=OFF を指定すると、編集マスクは使用されません。

ステートメントレベルで指定した編集マスクは、レポートレベルで指定した編集マスクを上書きします。

要素レベルで指定した編集マスクは、ステートメントレベルで指定した編集マスクを上書きします。

数値フィールドの編集マスク

数値フィールド（フォーマット N、I、P、F）の編集マスクは、数字（0 の場合も含む）で埋める出力位置ごとに 9 を含んでいる必要があります。

- 有効数字が 0 以外の場合にのみ出力位置を埋めるよう指示するには、Z を使用します。
- 小数点はピリオド（.）で示します。

小数点の右側に Z を指定することはできません。符号インジケータなどの先頭文字、末尾文字、および挿入文字は追加できます。

英数字フィールドの編集マスク

英数字フィールドの編集マスクは、出力する英数字ごとに X が含まれている必要があります。

いくつかの例外を除いて、先頭文字、末尾文字、および挿入文字は、アポストロフィで囲んでも囲まなくても、追加できます。

シルコンフレクス文字（^）は、編集マスクに空白を挿入するために、数値フィールドおよび英数字フィールドの両方で使用します。

フィールドの長さ

編集マスクを割り当てるフィールドの長さに注意することが重要です。

- 編集マスクがフィールドより長い場合、予期しない結果が発生します。
- 編集マスクがフィールドより短い場合、フィールド出力は、編集マスクに指定された桁数に切り詰められます。

例：

ある英数字フィールドの長さが 12 文字で、出力されるフィールド値が JOHNSON であると仮定した場合、編集マスクによって以下のような結果が得られます。

編集マスク	出力
EM=X.X.X.X.X	J.O.H.N.S
EM=****XXXXXX****	****JOHNSO**

日付／時刻フィールドの編集マスク

日付フィールドの編集マスクには、D（日）、M（月）、および Y（年）の文字をさまざまな組み合せで指定できます。

時刻フィールドの編集マスクには、H（時）、I（分）、S（秒）、および T（10 分の 1 秒）の文字をさまざまな組み合せで指定できます。

日付フィールドおよび時刻フィールドの編集マスクとともに、日時システム変数も参照してください。

セパレータ文字の表示のカスタマイズ

Natural プログラムは世界中のビジネスアプリケーションで使用されます。数値データフィールドおよび日時を含むフィールドを I/O ステートメントで表示する場合には、地域的な慣習に応じて、特別な出力スタイルで表示するのが一般的です。 それぞれの表示方法は、プログラムが実行されるロケールの機能として、選択的に処理される代替のプログラムコーディングで認識される必要はありません。ただし、小数点文字および 「千桁単位セパレータ文字」 を指定する一連のランタイムパラメータとともに、同じプログラムイメージを使用して実行される必要があります。

以下では次のトピックについて説明します。

- 小数点文字
- ダイナミック千桁単位セパレータ

- 例

小数点文字

Natural パラメータ DC（小数点文字）を使用すると、編集マスク内で小数点記号（「基数点文字」 とも呼ぶ）を表すために使用されている文字の代わりに挿入される文字を指定できます。このパラメータを使用すると、Naturalプログラムまたはアプリケーションのユーザーは、任意（特定）の文字を選択して、数値データ項目の整数部と小数部を分けることができます。例えば、米国の店舗では小数点（.）を使用し、ヨーロッパの店舗ではコンマ（,）を使用することなどができます。

ダイナミック千桁単位セパレータ

大きい整数値の出力を構築する場合、一般的には整数の3桁ごとにセパレータが挿入されて、千桁ごとに値が分割されます。 このセパレータは、「千桁単位セパレータ」と呼ばれます。 例えば、米国では通常、このためにコンマを使用しますが（1,000,000 など）、ドイツではピリオド（1.000.000）、フランスでは空白（1 000 000）を使用します。

Natural編集マスクでは、「ダイナミック千桁単位セパレータ」にコンマ（またはピリオド）を使用して、ランタイム時に千桁単位セパレータ文字（THSEPCHパラメータで定義）が挿入される位置を示します。 コンパイル時に、システムコマンド COMPOPT のオプション THSEP、プロファイルパラメータ CMPO のサブパラメータ THSEP、またはマクロ NTCCMPO を使用することで、コンマ（またはピリオド）をダイナミック千桁単位セパレータとして解釈可能または解釈不可にできます。

THSEPをOFFに設定した場合（デフォルト）、編集マスクで千桁単位セパレータとして使用されている文字はすべてリテラルとして処理され、ランタイム時にそのまま表示されます。 この設定は下位互換性を保持します。

THSEP を ON に設定した場合、編集マスク内のコンマ（またはピリオド）はすべてダイナミック千桁単位セパレータとして解釈されます。 一般的に、ダイナミック千桁単位セパレータはコンマですが、コンマがすでに小数点文字（DC）として使用されている場合には、ピリオドが千桁単位セパレータとして使用されます。

ランタイム時に、ダイナミック千桁単位セパレータは、THSEPCHパラメータの現在の値（千桁単位セパレータ文字）に置き換えられます。

例

パラメータ設定 DC='.' および THSEP=ON でカタログされている Natural プログラムで編集マスク（EM=ZZ,ZZZ,ZZ9.99）を使用します。

ランタイム時のパラメータ設定	表示内容
DC='.' および THSEPCH=','	1,234,567.89
DC=',' および THSEPCH='.'	1.234.567,89
DC=',' および THSEPCH='/'	1/234/567,89
DC=',' および THSEPCH=' '	1 234 567,89
DC=',' および THSEPCH='"'	1'234'567,89

編集マスクの例

編集マスクと編集マスクで作成される出力の例を以下に示します。

さらに、各編集マスクの省略表記法も示します。 省略形または長い表記のどちらでも使用できます。

編集マスク	省略形	出力 A	出力 B
EM=999.99	EM=9(3).9(2)	367.32	005.40
EM=ZZZZZ9	EM=Z(5)9(1)	0	579
EM=X^XXXXX	EM=X(1)^X(5)	B LUE	A 19379
EM=XXX...XX	EM=X(3)...X(2)	BLU...E	AAB...01
EM=MM.DD.YY	*	01.05.87	12.22.86
EM=HH.II.SS.T	**	08.54.12.7	14.32.54.3

* 日付システム変数を使用します。

** 時刻システム変数を使用します。

編集マスクの詳細については、『パラメータリファレンス』のセッションパラメータ EM を参照してください。

EM パラメータなしのプログラム例

```
** Example 'EDITMX01': Edit mask (using default edit masks)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 SALARY (1:3)
  2 CITY
END-DEFINE
*
READ (3) VIEWEMP BY NAME STARTING FROM 'JONES'
  DISPLAY 'N A M E'    NAME         /
          'OCCUPATION' JOB-TITLE
          'SALARY'     SALARY (1:3)
          'LOCATION'   CITY
  SKIP 1
END-READ
END
```

プログラム EDITMX01 の出力：

上記のプログラムの出力は、使用可能なデフォルトの編集マスクを示しています。

```
Page      1                                                 04-11-11  14:15:54

         N A M E            SALARY         LOCATION
       OCCUPATION
------------------------ ---------- --------------------

JONES                          46000 TULSA
MANAGER                        42300
                               39300

JONES                          50000 MOBILE
DIRECTOR                       46000
                               42700

JONES                          31000 MILWAUKEE
PROGRAMMER                     29400
                               27600
```

EM パラメータありのプログラム例

```
** Example 'EDITMX02': Edit mask (using EM)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 JOB-TITLE
  2 SALARY (1:3)
END-DEFINE
*
READ (3) VIEWEMP BY NAME STARTING FROM 'JONES'
 DISPLAY 'N A M E'    NAME         (EM=X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X) /
                       FIRST-NAME   (EM=...X(10)...)
         'OCCUPATION' JOB-TITLE    (EM=' ___ 'X(12))
         'SALARY'     SALARY (1:3) (EM=' USD 'ZZZ,999)
  SKIP 1
END-READ
END
```

プログラム EDITMX02 の出力：

前のプログラム（*EM パラメータなしのプログラム例*）の出力と比較して、EM 指定がフィールドの表示方法に与える影響を確認します。

```
Page      1                                                 04-11-11  14:15:54

           N A M E               OCCUPATION      SALARY
         FIRST-NAME
---------------------------- ---------------- -----------

J O N E S                     ___ MANAGER      USD  46,000
..VIRGINIA  ...                                USD  42,300
                                               USD  39,300

J O N E S                     ___ DIRECTOR     USD  50,000
..MARSHA    ...                                USD  46,000
                                               USD  42,700

J O N E S                     ___ PROGRAMMER   USD  31,000
..ROBERT    ...                                USD  29,400
                                               USD  27,600
```

編集マスクの他の例

次の例のプログラムを参照してください。

- ***EDITMX03 -** 編集マスク（英数字フィールドの異なる EM）*
- ***EDITMX04 -** 編集マスク（数値フィールドの異なる EM）*
- ***EDITMX05 -** 編集マスク（日付および時刻システム変数の EM）*

39 垂直表示

この章では、DISPLAY と WRITE ステートメントの機能を組み合わせて、フィールドの値を垂直表示する方法について説明します。

垂直表示の作成

垂直表示の作成には 2 つの方法があります。

- DISPLAY ステートメントと WRITE ステートメントの組み合わせを使用できます。
- DISPLAY ステートメントの VERT オプションを使用できます。

DISPLAY と WRITE の組み合わせ

「*DISPLAY および WRITE ステートメント*」で説明しているように、DISPLAY ステートメントは、通常、デフォルトヘッダー付きの列にデータを表示しますが、WRITE ステートメントはヘッダーなしでデータを横に表示します。

2 つのステートメントの機能を組み合わせて、フィールド値を垂直表示することができます。

DISPLAY ステートメントは、同じレコードの個々のフィールドの値を、フィールドごとに 1 列に、ページをまたがって出力します。 各レコードのフィールドの値は、前のレコードの値の下に表示されます。

DISPLAY ステートメントの後に WRITE ステートメントを使用することによって、WRITE ステートメントで指定したテキストやフィールド値を、DISPLAY ステートメントで表示されるレコードの間に挿入できます。

以下のプログラムは、DISPLAY と WRITE の組み合わせを示しています。

```
** Example 'WRITEX04': WRITE (in combination with DISPLAY)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 CITY
  2 DEPT
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'SAN FRANCISCO'
  DISPLAY NAME JOB-TITLE
  WRITE 22T 'DEPT:' DEPT
  SKIP 1
```

```
END-READ
END
```

プログラム WRITEX04 の出力：

```
Page      1                                                  04-11-11  14:15:55

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

KOLENCE              MANAGER
                     DEPT: TECH05

GOSDEN               ANALYST
                     DEPT: TECH10

WALLACE              SALES PERSON
                     DEPT: SALE20
```

タブ表記 - T*field

前の例において、フィールド DEPT の位置はタブ表記 *n*T で決定されます。この例の 20T の場合、画面の 20 桁目から表示が始まることを意味しています。

WRITE ステートメントで指定したフィールド値は、タブ表記T**field*を使用することによって、プログラムの最初の DISPLAY ステートメントで指定したフィールド値に自動的に揃えることができます。*field* は、前者のフィールドを揃える対象となる後者のフィールドの名前です。

以下のプログラムでは、WRITE ステートメントによる出力は、T*JOB-TITLE という表記を使用してフィールド JOB-TITLE に揃えられています。

```
** Example 'WRITEX05': WRITE  (in combination with DISPLAY)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 DEPT
  2 CITY
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'SAN FRANCISCO'
  DISPLAY NAME JOB-TITLE
  WRITE T*JOB-TITLE 'DEPT:' DEPT
```

```
  SKIP 1
END-READ
END
```

プログラム WRITEX05 の出力：

```
Page      1                                                  04-11-11  14:15:55

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

KOLENCE              MANAGER
                     DEPT: TECH05

GOSDEN               ANALYST
                     DEPT: TECH10

WALLACE              SALES PERSON
                     DEPT: SALE20
```

位置指定表記 x/y

DISPLAY および WRITE ステートメントを連続して使用し、WRITE ステートメントで複数行が出力される場合、*x*/*y*（数字スラッシュ数字）表記を使用して、表示する行および列を指定できます。 位置指定表記は、DISPLAY または WRITE ステートメントの次の要素を最後の出力の *x* 行下の *y* 桁目から開始します。

以下のプログラムは、この表記の使用法を示しています。

```
** Example 'WRITEX06': WRITE  (with n/n)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 MIDDLE-I
  2 ADDRESS-LINE (1:1)
  2 CITY
  2 ZIP
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'NEW YORK'
  DISPLAY 'NAME AND ADDRESS' NAME
  WRITE   1/5  FIRST-NAME
```

```
          1/30 MIDDLE-I
          2/5  ADDRESS-LINE (1:1)
          3/5  CITY
          3/30 ZIP /
END-READ
END
```

プログラム WRITEX06 の出力：

```
Page      1                                                  04-11-11  14:15:55

  NAME AND ADDRESS
--------------------

RUBIN
    SYLVIA                   L
    2003 SARAZEN PLACE
    NEW YORK                 10036

WALLACE
    MARY                     P
    12248 LAUREL GLADE C
    NEW YORK                 10036

KELLOGG
    HENRIETTA                S
    1001 JEFF RYAN DR.
    NEWARK                   19711
```

DISPLAY VERT ステートメント

Natural の標準の表示モードは水平方向です。

DISPLAY ステートメントの VERT 節オプションを使用すると、標準表示を上書きして、フィールドを垂直表示できます。

同じ DISPLAY ステートメントで使用できる HORIZ 節オプションは、標準の水平表示モードを再度有効にします。

垂直モードの列ヘッダーは、AS 節のさまざまなフォームで制御されます。 以下のプログラム例は、DISPLAY VERT ステートメントの使用法を示しています。

- AS 節のない DISPLAY VERT
- DISPLAY VERT AS CAPTIONED および HORIZ
- DISPLAY VERT AS 'text'
- DISPLAY VERT AS 'text' CAPTIONED

- タブ表記 P*field

AS 節のない DISPLAY VERT

以下のプログラムでは AS 節を使用していません。つまり、列ヘッダーは出力されません。

```
** Example 'DISPLX09': DISPLAY  (without column title)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'NEW YORK'
  DISPLAY VERT NAME FIRST-NAME / CITY
  SKIP 2
END-READ
END
```

プログラム DISPLX09 の出力：

すべてのフィールド値が垂直方向に表示されることに注意してください。

```
Page      1                                                 04-11-11  14:15:54


RUBIN
SYLVIA

NEW YORK


WALLACE
MARY

NEW YORK


KELLOGG
HENRIETTA

NEWARK
```

DISPLAY VERT AS CAPTIONED および HORIZ

以下のプログラムには VERT および HORIZ 節が含まれており、一部の列の値が垂直方向に出力され、その他の値が水平方向に出力されます。さらに、AS CAPTIONEDによってデフォルトの列ヘッダーが表示されます。

```
** Example 'DISPLX10': DISPLAY (with VERT as CAPTIONED and HORIZ clause)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
  2 SALARY (1:1)
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'NEW YORK'
  DISPLAY VERT AS CAPTIONED NAME FIRST-NAME
          HORIZ JOB-TITLE SALARY (1:1)
  SKIP 1
END-READ
END
```

プログラム DISPLX10 の出力：

```
Page      1                                                 04-11-11  14:15:54

        NAME                   CURRENT             ANNUAL
     FIRST-NAME               POSITION             SALARY
-------------------- ------------------------- ----------

RUBIN                SECRETARY                        17000
SYLVIA

WALLACE              ANALYST                          38000
MARY

KELLOGG              DIRECTOR                         52000
HENRIETTA
```

DISPLAY VERT AS 'text'

以下のプログラムには AS '*text*' 節が含まれており、指定した '*text*' が列ヘッダーとして表示されます。

注意: DISPLAY ステートメントのテキスト要素内のスラッシュ（/）は行送りを発生させます。

```
** Example 'DISPLX11': DISPLAY  (with VERT AS 'text' clause)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
  2 SALARY (1:1)
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'NEW YORK'
  DISPLAY VERT AS 'EMPLOYEES' NAME FIRST-NAME
          HORIZ JOB-TITLE SALARY (1:1)
  SKIP 1
END-READ
END
```

プログラム DISPLX11 の出力：

```
Page      1                                                 04-11-11  14:15:54

     EMPLOYEES                CURRENT            ANNUAL
                             POSITION            SALARY
-------------------- ------------------------- ----------

RUBIN                SECRETARY                      17000
SYLVIA

WALLACE              ANALYST                        38000
MARY

KELLOGG              DIRECTOR                       52000
HENRIETTA
```

DISPLAY VERT AS 'text' CAPTIONED

AS '*text*' CAPTIONED 節によって指定したテキストが列ヘッダーとして表示され、出力する各行のフィールド値の直前にデフォルトの列ヘッダーが表示されます。

以下のプログラムには、AS '*text*' CAPTIONED 節が含まれています。

```
** Example 'DISPLX12': DISPLAY (with VERT AS 'text' CAPTIONED clause)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
  2 SALARY (1:1)
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'NEW YORK'
  DISPLAY VERT AS 'EMPLOYEES' CAPTIONED NAME FIRST-NAME
          HORIZ JOB-TITLE SALARY (1:1)
  SKIP 1
END-READ
END
```

プログラム DISPLX12 の出力：

この節によって、デフォルトの列ヘッダー（NAME および FIRST-NAME）がフィールド値の前に出力されます。

```
Page      1                                              04-11-11  14:15:54

           EMPLOYEES                    CURRENT            ANNUAL
                                       POSITION            SALARY
------------------------------ ------------------------- ----------

NAME RUBIN                    SECRETARY                      17000
FIRST-NAME SYLVIA

NAME WALLACE                  ANALYST                        38000
FIRST-NAME MARY

NAME KELLOGG                  DIRECTOR                       52000
FIRST-NAME HENRIETTA
```

タブ表記 P*field

DISPLAY VERT ステートメントと後続の WRITE ステートメントを組み合わせて使用する場合、WRITE ステートメントでタブ表記 P*field-name を使用して、フィールド位置を DISPLAY VERT ステートメントで指定した特定のフィールドの列および行の位置に揃えることができます。

以下のプログラムでは、フィールド SALARY および BONUS が同じ列に表示され、1 番目の各行に SALARY、2 番目の各行に BONUS が表示されます。 テキスト ***SALARY PLUS BONUS*** は SALARY に揃えられます。つまり、テキストは SALARY と同じ列の 1 番目の行に表示されます。一方、テキスト (IN US DOLLARS) は BONUS に揃えられるため、BONUS と同じ列の 2 番目の行に表示されます。

```
** Example 'WRITEX07': WRITE  (with P*field)
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 JOB-TITLE
  2 SALARY (1:1)
  2 BONUS  (1:1,1:1)
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'LOS ANGELES'
  DISPLAY NAME JOB-TITLE
          VERT AS 'INCOME' SALARY (1) BONUS (1,1)
  WRITE P*SALARY '***SALARY PLUS BONUS***'
        P*BONUS  '(IN US DOLLARS)'
  SKIP 1
END-READ
END
```

プログラム WRITEX07 の出力：

```
Page      1                                                  04-11-11  14:15:55

        NAME                  CURRENT            INCOME
                             POSITION
-------------------- ------------------------- ----------

SMITH                                                   0
                                                        0
                                                ***SALARY PLUS BONUS***
                                                (IN US DOLLARS)

POORE JR             SECRETARY                      25000
                                                        0
                                                ***SALARY PLUS BONUS***
```

```
                                          (IN US DOLLARS)

PREPARATA            MANAGER                   46000
                                                9000
                                          ***SALARY PLUS BONUS***
                                          (IN US DOLLARS)
```

DISPLAY VERT と WRITE ステートメントの他の例

次の例のプログラムを参照してください。

- ***WRITEX10 - WRITE（nT、T*field、および P*field を使用）***

40 プログラミングのその他のポイント

- ステートメント、プログラム、またはアプリケーションの終了
- 条件付き処理 **- IF** ステートメント
- ループ処理
- コントロールブレイク
- データ計算
- システム変数とシステム関数
- スタック
- 日付情報の処理
- 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
- コンパイルのポイント

41 ステートメント、プログラム、またはアプリケーションの終了

ステートメントの終了

ステートメントの終了を明示的に示すには、セミコロン（;）をステートメントとその次のステートメントの間に記述します。 これにより、プログラム構造をより明確にできますが、必須ではありません。

プログラムの終了

END ステートメントは、Natural のプログラム、サブプログラム、外部サブルーチン、またはヘルプルーチンの終了を示すために使用します。

これらの各オブジェクトでは、最後のステートメントとして END ステートメントを使用する必要があります。

どのオブジェクトでも、END ステートメントを 1 つだけ使用します。

アプリケーションの終了

STOP ステートメントによる、アプリケーションの実行の終了

STOP ステートメントは、Natural アプリケーションの実行を終了するために使用します。 アプリケーション内の任意の場所で STOP ステートメントを実行すると、アプリケーション全体の実行が即座に停止されます。

TERMINATE ステートメントによる、アプリケーションの実行の終了

TERMINATE ステートメントを使用すると、Natural アプリケーションの実行が停止され、Natural セッションも終了します。

42 条件付き処理 - IF ステートメント

IF ステートメントで論理条件を定義します。IF ステートメントに付随するステートメントの実行は、その条件に依存します。

IF ステートメントの構造

IF ステートメントには、以下の 3 つの構成要素があります。

IF	IF 節には、満たす必要のある論理条件を指定します。
THEN	THEN 節には、この条件を満たしている場合に実行するステートメント（複数可）を指定します。
ELSE	（任意の）ELSE 節では、この条件を満たしていない場合に実行するステートメント（複数可）を指定できます。

したがって、IF ステートメントの一般的な形式は以下のようになります。

```
IFcondition
   THENexecute statement(s)
   ELSEexecute other statement(s)
END-IF
```

注意: IF 条件を満たしていないときにのみ特定の処理を実行する場合、THEN IGNORE 節を指定できます。 IGNORE ステートメントにより、条件を満たすと IF 条件は無視されます。

例 **1**：

```
** Example 'IFX01': IF
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 BIRTH
  2 CITY
  2 SALARY (1:1)
END-DEFINE
*
LIMIT 7
READ MYVIEW BY CITY STARTING FROM 'C'
 IF SALARY (1) LT 40000 THEN
    WRITE NOTITLE '*****' NAME 30X 'SALARY LT 40000'
  ELSE
    DISPLAY NAME BIRTH (EM=YYYY-MM-DD) SALARY (1)
  END-IF
END-READ
END
```

上記のプログラムの IF ステートメントブロックでは、以下の条件処理が実行されます。

- （IF）給与が 40000 未満の場合、（THEN）WRITE ステートメントが実行されます。
- それ以外の場合（ELSE）、つまり、給与が 40000 以上の場合は DISPLAY ステートメントが実行されます。

プログラム IFX01 の出力：

```
        NAME            DATE      ANNUAL
                         OF       SALARY
                       BIRTH
-------------------- ---------- ----------

***** KEEN                                          SALARY LT 40000
***** FORRESTER                                     SALARY LT 40000
***** JONES                                         SALARY LT 40000
***** MELKANOFF                                     SALARY LT 40000
DAVENPORT           1948-12-25      42000
GEORGES             1949-10-26     182800
***** FULLERTON                                     SALARY LT 40000
```

例 **2**：

```
** Example 'IFX03': IF
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 BONUS  (1,1)
  2 SALARY (1)
*
1 #INCOME  (N9)
1 #TEXT    (A26)
END-DEFINE
*
WRITE TITLE '-- DISTRIBUTION OF CATALOGS I AND II --' /
*
READ (3) EMPLOY-VIEW BY CITY = 'SAN FRANSISCO'
  COMPUTE #INCOME = BONUS(1,1) + SALARY(1)
  /*
 IF #INCOME > 40000
    MOVE 'CATALOGS I AND II' TO #TEXT
  ELSE
    MOVE 'CATALOG I'         TO #TEXT
  END-IF
  /*
  DISPLAY NAME 5X 'SALARY' SALARY(1) / BONUS(1,1)
```

```
  WRITE T*SALARY '-'(10) /
        16X 'INCOME:' T*SALARY #INCOME 3X #TEXT /
        16X '='(19)
  SKIP 1
END-READ
END
```

プログラム IFX03 の出力：

```
                    -- DISTRIBUTION OF CATALOGS I AND II --

        NAME               SALARY
                           BONUS
--------------------     ----------

COLVILLE JR                   56000
                                  0
                           ----------
                INCOME:       56000   CATALOGS I AND II
                ====================

RICHMOND                       9150
                                  0
                           ----------
                INCOME:        9150   CATALOG I
                ====================

MONKTON                       13500
                                600
                           ----------
                INCOME:       14100   CATALOG I
                ====================
```

IF ステートメントのネスト

IF ステートメントは、複数のネスト構造にして使用できます。例えば、THEN 節の実行を、THEN 節に指定した別の IF ステートメントに依存させることができます。

例：

```
** Example 'IFX02': IF (two IF statements nested)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY (1:1)
  2 BIRTH
  2 PERSONNEL-ID
1 MYVIEW2 VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
*
1 #BIRTH   (D)
END-DEFINE
*
MOVE EDITED '19450101' TO #BIRTH (EM=YYYYMMDD)
*
LIMIT 20
FND1. FIND MYVIEW WITH CITY = 'BOSTON'
                  SORTED BY NAME
 IF SALARY (1) LESS THAN 20000
    WRITE NOTITLE '*****' NAME 30X 'SALARY LT 20000'
  ELSE
    IF BIRTH GT #BIRTH
      FIND MYVIEW2 WITH PERSONNEL-ID = PERSONNEL-ID (FND1.)
        DISPLAY (IS=ON) NAME BIRTH (EM=YYYY-MM-DD)
                        SALARY (1) MAKE (AL=8 IS=OFF)
      END-FIND
    END-IF
  END-IF
  SKIP 1
END-FIND
END
```

プログラム IFX02 の出力：

```
        NAME            DATE      ANNUAL     MAKE
                         OF       SALARY
                       BIRTH
-------------------- ---------- ---------- --------

***** COHEN                                          SALARY LT 20000

CREMER               1972-12-14      20000 FORD

***** FLEMING                                        SALARY LT 20000
```

```
PERREAULT            1950-05-12      30500 CHRYSLER

***** SHAW                                                SALARY LT 20000

STANWOOD             1946-09-08      31000 CHRYSLER
                                           FORD
```

43 ループ処理

記述した条件が満たされるまで、または一定の条件が有効である限り、処理ループはグループ化されたステートメントを繰り返し実行します。

処理ループの使用

処理ループはデータベースループと非データベースループに分類できます。

- データベース処理ループ
  READ、FIND、または HISTOGRAM の各ステートメントの結果としてデータベースから取得したデータを処理するために、Naturalによって自動的に作成されます。これらのステートメントの詳細については、「*Adabas データベースのデータへのアクセス*」を参照してください。
- 非データベース処理ループ
  ステートメント REPEAT、FOR、CALL FILE、CALL LOOP、SORT、および READ WORK FILE によって開始されます。

複数の処理ループを同時にアクティブにできます。 ループは、アクティブな（開いている）他のループ内に埋め込んだり、ネストしたりできます。

処理ループは、対応する END-... ステートメント（END-REPEAT、END-FOR など）で明示的に閉じる必要があります。

オペレーティングシステムのソートプログラムを呼び出す SORT ステートメントは、すべてのアクティブな処理ループを閉じて、新しい処理ループを開始します。

データベースループの制限

以下では次のトピックについて説明します。

- 有効なデータベースループの制限方法
- LT セッションパラメータ
- LIMIT ステートメント
- リミット表記

- 制限設定の優先順位

有効なデータベースループの制限方法

ステートメント READ、FIND、または HISTOGRAM で開始した処理ループの繰り返し回数を制限するには、以下の 3 つの方法があります。

- セッションパラメータ LT を使用します。
- LIMIT ステートメントを使用します。
- または、READ/FIND/HISTOGRAM ステートメント自体のリミット表記を使用します。

LT セッションパラメータ

システムコマンド GLOBALS には、セッションパラメータ LT を指定できます。このパラメータを使用して、データベース処理ループで読み込むレコード件数を制限します。

例：

```
GLOBALS LT=100
```

この制限は、セッション全体のすべての READ、FIND、および HISTOGRAM の各ステートメントに適用されます。

LIMIT ステートメント

プログラムでは、LIMIT ステートメントを使用して、データベース処理ループで読み込むレコード件数を制限できます。

例：

```
LIMIT 100
```

別の LIMIT ステートメントまたはリミット表記によって上書きされるまで、LIMIT ステートメントはプログラムの残りの部分に適用されます。

リミット表記

READ、FIND、または HISTOGRAM の各ステートメント自体で、読み込むレコード件数をステートメント名の直後のカッコ内に指定できます。

例：

```
READ (10) VIEWXYZ BY NAME
```

このリミット表記は、その他の有効なあらゆる制限を上書きします。ただし、リミット表記が適用されるのは、その表記が指定されているステートメントのみです。

制限設定の優先順位

LT パラメータに設定されている制限が、LIMIT ステートメントに指定されている制限またはリミット表記よりも小さい場合、LT の制限はこれらの他の制限より優先されます。

非データベースループの制限 - REPEAT ステートメント

非データベース処理ループは、論理条件基準、または他に指定された制限条件に基づいて開始および終了します。

ここでは、非データベースループステートメントとして、REPEAT ステートメントについて説明します。

REPEAT ステートメントを使用して、繰り返し実行する1つ以上のステートメントを指定します。さらに、ある条件に一致するまで、または、ある条件に一致している間にのみステートメントが実行されるように、論理条件を指定できます。 この目的のためには、UNTIL 節または WHILE 節を使用します。

論理条件を指定する場合、以下のようにループは制御されます。

- UNTIL 節を使用すると、論理条件が満たされるまで REPEAT ループが続けられます。
- WHILE 節を使用すると、論理条件が真*である限り* REPEAT ループが続けられます。

論理条件を指定しない場合、以下のステートメントのいずれかを使用して、REPEAT ループを抜ける必要があります。

- ESCAPE：処理ループの実行を終了し、ループの外側の処理を継続します（下記参照）。
- STOP：Natural アプリケーション全体の実行を停止します。
- TERMINATE：Natural アプリケーションの実行を停止し、Natural セッションも終了します。

REPEAT ステートメントの例

```
** Example 'REPEAX01': REPEAT
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 SALARY (1:1)
*
1 #PAY1    (N8)
END-DEFINE
*
READ (5) MYVIEW BY NAME WHERE SALARY (1) = 30000 THRU 39999
  MOVE SALARY (1) TO #PAY1
  /*
  REPEAT WHILE #PAY1 LT 40000
    MULTIPLY #PAY1 BY 1.1
    DISPLAY NAME (IS=ON) SALARY (1)(IS=ON) #PAY1
  END-REPEAT
  /*
  SKIP 1
END-READ
END
```

プログラム REPEAX01 の出力：

```
Page      1                                                 04-11-11  14:15:54

        NAME           ANNUAL     #PAY1
                       SALARY
-------------------- ---------- ---------

ADKINSON                  34500     37950
                                    41745

                          33500     36850
                                    40535

                          36000     39600
                                    43560

AFANASSIEV                37000     40700

ALEXANDER                 34500     37950
                                    41745
```

処理ループの終了 - ESCAPE ステートメント

ESCAPE ステートメントは、論理条件に基づいて処理ループの実行を終了するために使用します。

ESCAPE ステートメントは、IF 条件ステートメントグループのループ内、およびブレイク処理ステートメントグループ（AT END OF DATA、AT END OF PAGE、AT BREAK）のループ内で指定できます。また、非データベースループの基本論理条件を実装している独立操作可能なステートメントとして指定できます。

ESCAPE ステートメントには、ESCAPE ステートメントで処理ループを抜けた後にどこから処理を継続するのかを指定するオプション TOP と BOTTOM が用意されています。

- ESCAPE TOP は、処理ループの先頭から処理を継続するために使用します。
- ESCAPE BOTTOM は、処理ループの後の最初のステートメントから処理を継続するために使用します。

同じ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 ステートメントの詳細と例については、『ステートメント』ドキュメントを参照してください。

ループ内のループ

別のデータベースステートメントによって開始されたデータベース処理ループ内にデータベースステートメントを指定できます。 データベースループ開始ステートメントがこの方法で埋め込まれていると、ループの 「階層」 が作成され、階層ごとに、選択条件を満たす各レコードが処理されます。

複数のレベルのループを埋め込むことができます。例えば、非データベースループをデータベースループ内にネストできます。 また、データベースループを非データベースループ内にネストすることもできます。 データベースループおよび非データベースループは、条件付きステートメントグループ内でネストできます。

FIND ステートメントのネストの例

以下のプログラムは、2 つのループ階層を示しています。1 つの FIND ループ内に別の FIND ループがネスト、つまり埋め込まれています。

```
** Example 'FINDX06': FIND (two FIND statements nested)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 PERSONNEL-ID
1 VEH-VIEW VIEW OF VEHICLES
  2 MAKE
  2 PERSONNEL-ID
END-DEFINE
*
FND1. FIND EMPLOY-VIEW WITH CITY = 'NEW YORK' OR = 'BEVERLEY HILLS'
  FIND (1) VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (FND1.)
    DISPLAY NOTITLE NAME CITY MAKE
  END-FIND
END-FIND
END
```

上記のプログラムでは、複数のファイルからデータを選択しています。 外側の FIND ループでは、EMPLOYEES ファイルから、ニューヨークまたはビバリーヒルズに住んでいるすべての個人を選択しています。 内側の FIND ループでは、外側のループで選択されたレコードごとに、VEHICLES ファイルからその個人の車のデータを選択しています。

プログラム FINDX06 の出力：

```
        NAME                 CITY                 MAKE
-------------------- -------------------- --------------------

RUBIN                NEW YORK             FORD
OLLE                 BEVERLEY HILLS       GENERAL MOTORS
WALLACE              NEW YORK             MAZDA
JONES                BEVERLEY HILLS       FORD
SPEISER              BEVERLEY HILLS       GENERAL MOTORS
```

プログラム内のステートメント参照

ステートメント参照表記は、以下の目的で使用します。

- 特定の範囲のデータに対する処理を指定するために、プログラム内で前に処理したステートメントを参照するため。
- Natural のデフォルトの参照設定を上書きするため。
- コードをわかりやすくするため。

処理ループの開始、またはデータベースのデータ要素へのアクセスを発生させる、あらゆる Natural ステートメントを参照できます。 例えば、以下のステートメントを参照できます。

- READ
- FIND
- HISTOGRAM
- SORT
- REPEAT
- FOR

プログラムで複数の処理ループを使用するときは、参照表記を使用して、データベースフィールドに最初にアクセスしたステートメントを参照することにより、処理対象のデータベースフィールドを一意に指定します。

このような方法でフィールドを参照できるかどうかは、『ステートメント』ドキュメントに記載されている、対応するステートメントの説明の「*オペランド定義テーブル*」の ステートメント参照 列を参照してください。「ユーザー定義変数」の「表記 (*r*) を使用したデータベースフィールドの参照」を参照してください。

また、参照表記はいくつかのステートメントで指定できます。 次に例を示します。

- AT START OF DATA
- AT END OF DATA
- AT BREAK
- ESCAPE BOTTOM

参照表記を使用しないと、AT START OF DATA、AT END OF DATA、または AT BREAK ステートメントは、*最も外側の*アクティブな READ、FIND、HISTOGRAM、SORT、または READ WORK FILE の各ループに関連付けられます。 参照表記を使用すると、別のアクティブな処理ループに関連付けることができます。

参照表記を ESCAPE BOTTOM ステートメントに指定すると、参照表記で指定した処理ループの後の最初のステートメントから処理が継続されます。

ステートメント参照表記は、*ステートメント参照ラベル*形式または*ソースコード行番号*形式で指定できます。

- ステートメント参照ラベル
  ステートメント参照ラベルはいくつかの文字で構成され、最後には必ずピリオド（.）を使用します。 ピリオドによって、そのエントリはラベルとして識別されます。

  ステートメントが含まれる行の先頭にラベルを指定することにより、参照されるステートメントをラベルでマークします。 次に例を示します。

```
0030 ...
0040 READ1. READ VIEWXYZ BY NAME
0050 ...
```

  マークされたステートメントを参照するステートメントでは、ステートメントの構文図に示されている位置に、ラベルをカッコで囲んで指定します（『ステートメント』ドキュメントを参照）。 次に例を示します。

```
AT BREAK (READ1.) OF NAME
```

- ソースコード行番号
  ソースコード行番号を使用して参照する場合、行番号は、カッコで囲んだ4桁の数字（先行ゼロは省略不可）で指定する必要があります。 次に例を示します。

```
AT BREAK (0040) OF NAME
```

  ステートメント内でラベル／行番号を使用して特定のフィールドを前のステートメントに関連付ける場合、ラベル／行番号は、フィールド名の後にカッコで囲んで指定します。 次に例を示します。

```
DISPLAY NAME (READ1.) JOB-TITLE (READ1.) MAKE MODEL
```

行番号とラベルは区別なく使用できます。

「ユーザー定義変数」の「表記 (*r*) を使用したデータベースフィールドの参照」を参照してください。

行番号を使用した参照の例

以下のプログラムでは、参照にソースコード行番号（カッコで囲んだ4桁の数字）を使用しています。

この例では、参照先のステートメントへのすべての参照において、デフォルトで行番号が使用されています。

```
0010 ** Example 'LABELX01': Labels for READ and FIND loops (line numbers)
0020 ************************************************************************
0030 DEFINE DATA LOCAL
0040 1 MYVIEW1 VIEW OF EMPLOYEES
0050   2 NAME
0060  2 FIRST-NAME
0070  2 PERSONNEL-ID
0080 1 MYVIEW2 VIEW OF VEHICLES
0090  2 PERSONNEL-ID
0100  2 MAKE
0110 END-DEFINE
0120 *
0130 LIMIT 15
0140 READ MYVIEW1 BY NAME STARTING FROM 'JONES'
0150  FIND MYVIEW2 WITH PERSONNEL-ID = PERSONNEL-ID (0140)
0160    IF NO RECORDS FOUND
0170      MOVE '***NO CAR***' TO MAKE
0180    END-NOREC
0190    DISPLAY NOTITLE NAME       (0140) (IS=ON)
0200                    FIRST-NAME (0140) (IS=ON)
0210                    MAKE       (0150)
0220  END-FIND /* (0150)
0230 END-READ   /* (0140)
0240 END
```

ステートメント参照ラベルを使用した参照の例

以下の例は、ステートメント参照ラベルの使用方法を示しています。

行番号の代わりにラベルが参照に使用されている点以外は、前述のプログラムと同一です。

```
** Example 'LABELX02': Labels for READ and FIND loops (user labels)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW1 VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
1 MYVIEW2 VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
RD. READ MYVIEW1 BY NAME STARTING FROM 'JONES'
  FD. FIND MYVIEW2 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       (RD.)
  END-FIND /* (RD.)
END-READ   /* (RD.)
END
```

どちらのプログラムでも、以下の出力が生成されます。

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

JONES                VIRGINIA             CHRYSLER
                     MARSHA               CHRYSLER
                                          CHRYSLER
                     ROBERT               GENERAL MOTORS
                     LILLY                FORD
                                          MG
                     EDWARD               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***
KANT                 HEIKE                ***NO CAR***
```

44 コントロールブレイク

この章では、ステートメントの実行をコントロールブレイクに依存させる方法、およびNaturalシステム関数の評価にコントロールブレイクを使用する方法について説明します。

コントロールブレイクの使用

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と発生します。

ステートメントの実行をコントロールブレイクに依存させることができます。

また、Natural システム関数の評価にコントロールブレイクを使用することもできます。

システム関数については、「*システム変数とシステム関数*」で説明します。 有効な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。

AT BREAK ステートメント

AT BREAK ステートメントでは、コントロールブレイクが起こるたび、つまり、AT BREAK ステートメントに指定したコントロールフィールドの値が変わるたびに実行する処理を指定します。コントロールフィールドとして、データベースフィールドまたはユーザー定義変数を使用できます。

以下では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**
- **複数のコントロールブレイクレベル**

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

AT BREAK ステートメントにコントロールフィールドとして指定されるフィールドは、通常はデータベースフィールドです。

例：

```
...
AT BREAK OF DEPT
  statements
END-BREAK
...
```

この例では、コントロールフィールドはデータベースフィールド DEPT です。このフィールドの値が変わると（例：SALE01 から SALE02 へ）、AT BREAK ステートメントに指定した *statements* が実行されます。

フィールド全体ではなく、フィールドの一部のみ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。 */n/* 表記を使用して、フィールドの先頭の *n* 桁のみを使用して値の変更をチェックするように指定できます。

例：

```
...
AT BREAK OF DEPT /4/
  statements
END-BREAK
...
```

この例では、指定した *statements* は、フィールド DEPT の先頭 4 桁の値が変わった場合（例：SALE から TECH へ）にのみ実行されます。ただし、フィールド値が SALE01 から SALE02 に変わっても、この変更は無視され、AT BREAK 処理は実行されません。

例：

```
** Example 'ATBREX01': AT BREAK OF (with database field)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 COUNTRY
  2 JOB-TITLE
  2 SALARY (1:1)
END-DEFINE
*
READ (5) MYVIEW BY CITY WHERE COUNTRY = 'USA'
  DISPLAY CITY (AL=9) NAME 'POSITION' JOB-TITLE 'SALARY' SALARY(1)
  /*
  AT BREAK OF CITY
    WRITE /   OLD(CITY) (EM=X^X^X^X^X^X^X^X^X^X^X^X^)
          5X 'AVERAGE:' T*SALARY AVER(SALARY(1)) //
              COUNT(SALARY(1)) 'RECORDS FOUND' /
  END-BREAK
  /*
  AT END OF DATA
    WRITE 'TOTAL (ALL RECORDS):' T*SALARY(1) TOTAL(SALARY(1))
  END-ENDDATA
END-READ
END
```

上記のプログラムでは、フィールド CITY の値が変わるたびに、最初の WRITE ステートメントが実行されます。

AT BREAK ステートメントでは、Natural システム関数 OLD、AVER、および COUNT が評価され、その結果が WRITE ステートメントで出力されます。

AT END OF DATA ステートメントでは、Natural システム関数 TOTAL が評価されます。

プログラム ATBREX01 の出力：

```
Page      1                                                  04-12-14  14:07:26

  CITY            NAME                 POSITION            SALARY
--------- -------------------- ------------------------ ----------

AIKEN     SENKO                PROGRAMMER                    31500

A I K E N                   AVERAGE:                        31500

       1 RECORDS FOUND

ALBUQUERQ HAMMOND              SECRETARY                     22000
ALBUQUERQ ROLLING              MANAGER                       34000
ALBUQUERQ FREEMAN              MANAGER                       34000
ALBUQUERQ LINCOLN              ANALYST                       41000

A L B U Q U E R Q U E      AVERAGE:                          32750

       4 RECORDS FOUND

TOTAL (ALL RECORDS):                                      162500
```

ユーザー定義変数に基づくコントロールブレイク

ユーザー定義変数を AT BREAK ステートメントの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。

以下のプログラムでは、ユーザー定義変数 #LOCATION が、コントロールフィールドとして使用されています。

```
** Example 'ATBREX02': AT BREAK OF  (with user-defined variable and
**                      in conjunction with BEFORE BREAK PROCESSING)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
  2 JOB-TITLE
  2 SALARY (1:1)
*
1 #LOCATION (A20)
END-DEFINE
*
READ (5) MYVIEW BY CITY WHERE COUNTRY = 'USA'
```

```
  BEFORE BREAK PROCESSING
    COMPRESS CITY 'USA' INTO #LOCATION
  END-BEFORE
  DISPLAY #LOCATION 'POSITION' JOB-TITLE 'SALARY' SALARY (1)
  /*
  AT BREAK OF #LOCATION
    SKIP 1
  END-BREAK
END-READ
END
```

プログラム ATBREX02 の出力：

```
Page      1                                                 04-12-14  14:08:36

     #LOCATION               POSITION            SALARY
-------------------- ------------------------- ----------

AIKEN USA            PROGRAMMER                   31500

ALBUQUERQUE USA      SECRETARY                    22000
ALBUQUERQUE USA      MANAGER                      34000
ALBUQUERQUE USA      MANAGER                      34000
ALBUQUERQUE USA      ANALYST                      41000
```

複数のコントロールブレイクレベル

上記の説明のとおり、*/n/* 表記を使用すると、フィールドの一部分をコントロールブレイクのためにチェックできます。 フィールド全体をあるブレイクのコントロールフィールドとして使用し、同じフィールドの一部を別のブレイクのコントロールフィールドとして使用することにより、複数の AT BREAK ステートメントを組み合わせることができます。

このような場合、下位レベル（フィールド全体）のブレイクは、上位レベル（フィールドの一部）のブレイクの前に指定する必要があります。つまり、最初の AT BREAK ステートメントでフィールド全体をコントロールフィールドとして指定し、2番目のステートメントでフィールドの一部をコントロールフィールドとして指定する必要があります。

以下のプログラム例は、フィールド DEPT とその最初の 4 桁（DEPT /4/）を使用してこれを説明したものです。

```
** Example 'ATBREX03': AT BREAK OF (two statements in combination)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 DEPT
  2 SALARY    (1:1)
  2 CURR-CODE (1:1)
END-DEFINE
*
READ MYVIEW BY DEPT STARTING FROM 'SALE40' ENDING AT 'TECH10'
     WHERE SALARY(1) GT 47000 AND CURR-CODE(1) = 'USD'
  /*
  AT BREAK OF DEPT
    WRITE '*** LOWEST BREAK LEVEL ***' /
  END-BREAK
  AT BREAK OF DEPT /4/
    WRITE '*** HIGHEST BREAK LEVEL ***'
  END-BREAK
  /*
  DISPLAY DEPT NAME 'POSITION' JOB-TITLE
END-READ
END
```

プログラム ATBREX03 の出力：

```
Page      1                                                 04-12-14  14:09:20

DEPARTMENT         NAME                 POSITION
   CODE
---------- -------------------- -------------------------

TECH05     HERZOG               MANAGER
TECH05     LAWLER               MANAGER
TECH05     MEYER                MANAGER
*** LOWEST BREAK LEVEL ***

TECH10     DEKKER               DBA
*** LOWEST BREAK LEVEL ***

*** HIGHEST BREAK LEVEL ***
```

以下のプログラムでは、フィールド DEPT の値が変わるたびに 1 行の空行が出力されます。また、DEPT の先頭 4 桁の値が変わるたびにシステム関数 COUNT が評価され、レコード件数が算出されます。

```
** Example 'ATBREX04': AT BREAK OF (two statements in combination)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 DEPT
  2 REDEFINE DEPT
    3 #GENDEP (A4)
  2 NAME
  2 SALARY    (1)
END-DEFINE
*
WRITE TITLE '** PERSONS WITH SALARY > 30000, SORTED BY DEPARTMENT **' /
LIMIT 9
READ MYVIEW BY DEPT FROM 'A' WHERE SALARY(1) > 30000
  DISPLAY 'DEPT' DEPT NAME 'SALARY' SALARY(1)
  /*
  AT BREAK OF DEPT
    SKIP 1
  END-BREAK
  AT BREAK OF DEPT /4/
    WRITE COUNT(SALARY(1)) 'RECORDS FOUND IN:' OLD(#GENDEP) /
  END-BREAK
END-READ
END
```

プログラム ATBREX04 の出力：

```
            ** PERSONS WITH SALARY > 30000, SORTED BY DEPARTMENT **

 DEPT          NAME           SALARY
------ -------------------- ----------

ADMA01 JENSEN                  180000
ADMA01 PETERSEN                105000
ADMA01 MORTENSEN               320000
ADMA01 MADSEN                  149000
ADMA01 BUHL                    642000

ADMA02 HERMANSEN               391500
ADMA02 PLOUG                   162900
ADMA02 HANSEN                  234000

       8 RECORDS FOUND IN: ADMA

COMP01 HEURTEBISE              168800
```

```
      1 RECORDS FOUND IN: COMP
```

自動ブレイク処理

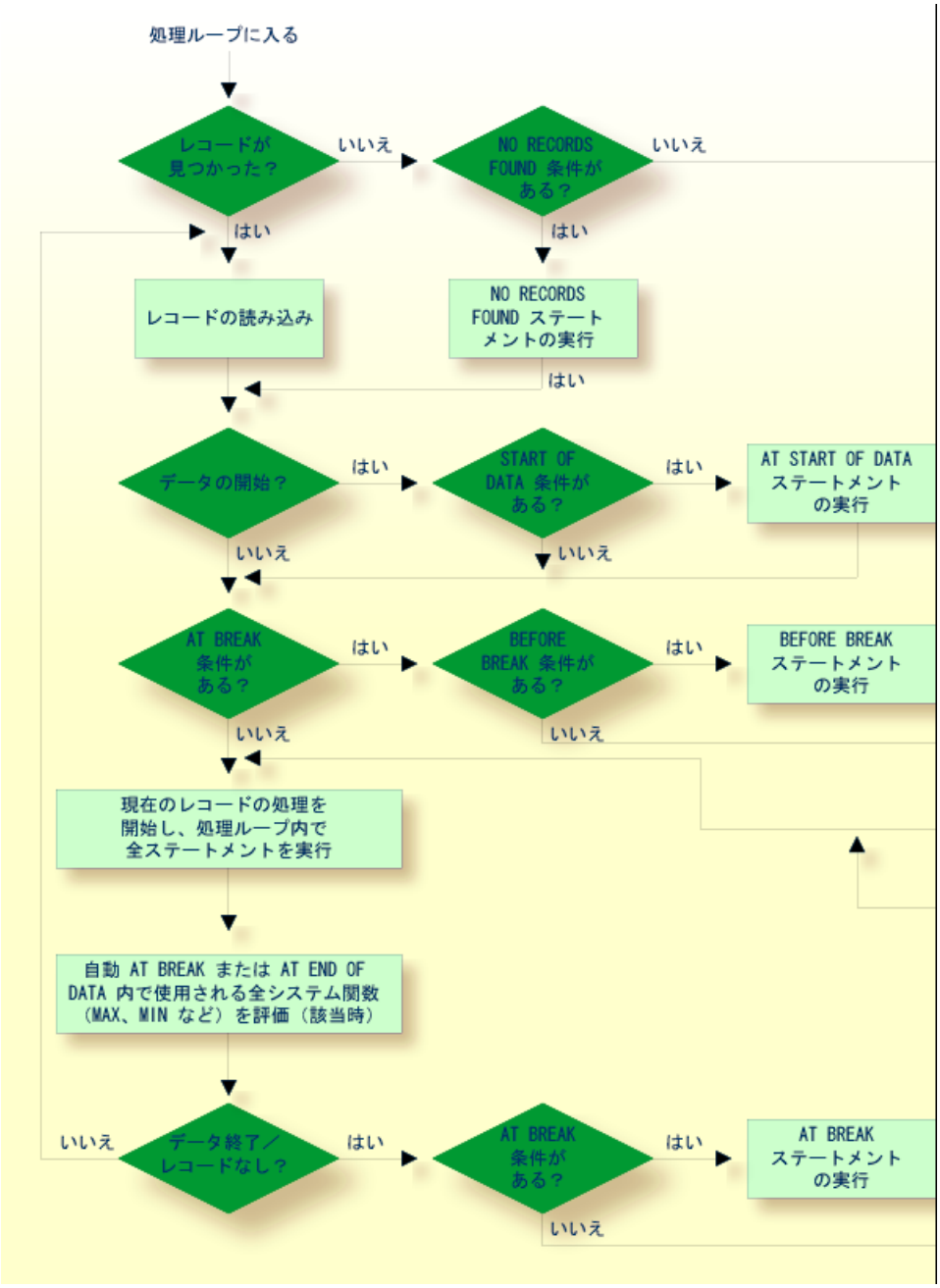
自動ブレイク処理は、AT BREAK ステートメントが含まれている処理ループで実行されます。 これは以下のステートメントに適用されます。

- FIND
- READ
- HISTOGRAM
- SORT
- READ WORK FILE

AT BREAK ステートメントに指定されているコントロールフィールドの値は、WITH 節および WHERE 節の選択条件を満たしているレコードに対してのみチェックされます。

Natural システム関数（AVER、MAX、MIN など）は、処理ループ内のすべてのステートメントを実行した後に、レコードごとに評価されます。 システム関数は、WHERE 条件で拒否されたレコードに対しては評価されません。

下の図は、自動ブレイク処理のフローロジックを示しています。


処理ループに入る
レコードが見つかった？
いいえ
NO RECORDS FOUND 条件がある？
いいえ
はい
はい
レコードの読み込み
NO RECORDS FOUND ステートメントの実行
はい
データの開始？
はい
START OF DATA 条件がある？
はい
AT START OF DATA ステートメントの実行
いいえ
いいえ
AT BREAK 条件がある？
はい
BEFORE BREAK 条件がある？
はい
BEFORE BREAK ステートメントの実行
いいえ
いいえ
現在のレコードの処理を開始し、処理ループ内で全ステートメントを実行
自動 AT BREAK または AT END OF DATA 内で使用される全システム関数（MAX、MIN など）を評価（該当時）
いいえ
データ終了／レコードなし？
はい
AT BREAK 条件がある？
はい
AT BREAK ステートメントの実行
いいえ

AT BREAK ステートメントとシステム関数の例

以下の例は、AT BREAK ステートメントにおける Natural システム関数 OLD、MIN、AVER、MAX、SUM、COUNT の使用方法（および AT END OF DATA ステートメントにおけるシステム関数 TOTAL の使用方法）を示しています。

```
** Example 'ATBREX05': AT BREAK OF (with system functions)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY    (1:1)
  2 CURR-CODE (1:1)
END-DEFINE
*
LIMIT 3
READ MYVIEW BY CITY = 'SALT LAKE CITY'
  DISPLAY NOTITLE CITY NAME 'SALARY' SALARY(1) 'CURRENCY' CURR-CODE(1)
  /*
  AT BREAK OF CITY
    WRITE /  OLD(CITY) (EM=X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X)
      31T ' - MINIMUM:' MIN(SALARY(1))  CURR-CODE(1) /
      31T ' - AVERAGE:' AVER(SALARY(1)) CURR-CODE(1) /
      31T ' - MAXIMUM:' MAX(SALARY(1))  CURR-CODE(1) /
      31T ' -     SUM:' SUM(SALARY(1))  CURR-CODE(1) /
      33T COUNT(SALARY(1)) 'RECORDS FOUND' /
  END-BREAK
  /*
  AT END OF DATA
    WRITE 22T 'TOTAL (ALL RECORDS):'
              T*SALARY TOTAL(SALARY(1)) CURR-CODE(1)
  END-ENDDATA
END-READ
END
```

プログラム ATBREX05 の出力：

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

SALT LAKE CITY       ANDERSON                  50000 USD
SALT LAKE CITY       SAMUELSON                 24000 USD

S A L T   L A K E   C I T Y    - MINIMUM:      24000 USD
                               - AVERAGE:      37000 USD
```

```
                                - MAXIMUM:      50000 USD
                                -     SUM:      74000 USD
                                        2 RECORDS FOUND

SAN DIEGO            GEE                        60000 USD

S A N   D I E G O               - MINIMUM:      60000 USD
                                - AVERAGE:      60000 USD
                                - MAXIMUM:      60000 USD
                                -     SUM:      60000 USD
                                        1 RECORDS FOUND

                     TOTAL (ALL RECORDS):     134000 USD
```

AT BREAK ステートメントの他の例

次の例のプログラムを参照してください。

- *ATBREX06 - AT BREAK OF（NMIN、NAVER、NCOUNT を MIN、AVER、COUNT と比較）*

BEFORE BREAK PROCESSING ステートメント

BEFORE BREAK PROCESSING ステートメントを使用すると、コントロールブレイクの直前、つまり、コントロールフィールドの値のチェック前、AT BREAK ブロックに指定したステートメントの実行前、および、あらゆるNaturalシステム関数の実行前に、実行するステートメントを指定できます。

BEFORE BREAK PROCESSING ステートメントの例

```
** Example 'BEFORX01': BEFORE BREAK PROCESSING
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 SALARY (1:1)
  2 BONUS  (1:1,1:1)
*
1 #INCOME  (P11)
END-DEFINE
*
```

```
LIMIT 5
READ MYVIEW BY NAME FROM 'B'
 BEFORE BREAK PROCESSING
    COMPUTE #INCOME = SALARY(1) + BONUS(1,1)
  END-BEFORE
  /*
  DISPLAY NOTITLE NAME FIRST-NAME (AL=10)
                  'ANNUAL/INCOME' #INCOME 'SALARY' SALARY(1) (LC==) /
                  '+ BONUS' BONUS(1,1) (IC=+)
  AT BREAK OF #INCOME
    WRITE T*#INCOME '-'(24)
  END-BREAK
END-READ
END
```

プログラム BEFORX01 の出力：

```
        NAME          FIRST-NAME    ANNUAL      SALARY
                                    INCOME      + BONUS
-------------------- ---------- ------------ -----------

BACHMANN             HANS              56800 =     52800
                                                   +4000
                                 ------------------------
BAECKER              JOHANNES          81000 =     74400
                                                   +6600
                                 ------------------------
BAECKER              KARL              52650 =     48600
                                                   +4050
                                 ------------------------
BAGAZJA              MARJAN           152700 =    129700
                                                  +23000
                                 ------------------------
BAILLET              PATRICK          198500 =    188000
                                                  +10500
                                 ------------------------
```

ユーザー開始のブレイク処理 - PERFORM BREAK PROCESSING ステートメント

自動ブレイク処理では、処理ループ内の AT BREAK ステートメントの位置に関係なく、指定したコントロールフィールドの値が変わるたびに、AT BREAK ブロックに指定したステートメントが実行されます。

PERFORM BREAK PROCESSING ステートメントを使用すると、処理ループ内の特定の位置でブレイク処理を実行できます。PERFORM BREAK PROCESSING ステートメントは、プログラムの処理フローの中で検出されると実行されます。

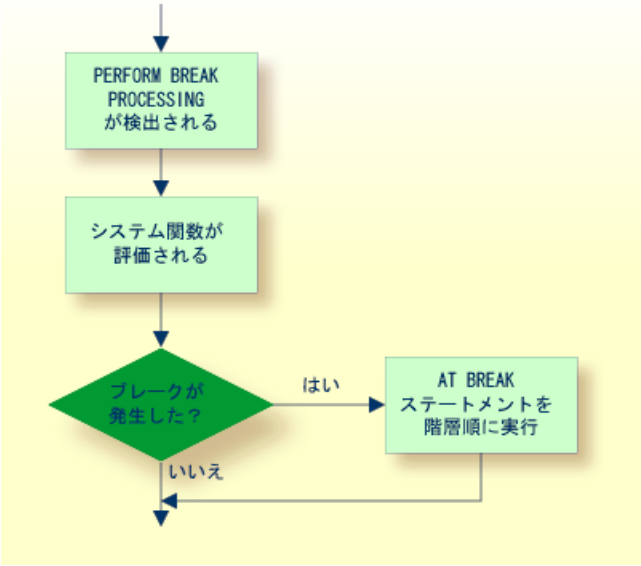
PERFORM BREAK PROCESSING の直後に、1 つ以上の AT BREAK ステートメントブロックを指定します。

```
...
PERFORM BREAK PROCESSING
  AT BREAK OF field1
    statements
  END-BREAK
  AT BREAK OF field2
    statements
  END-BREAK
...
```

PERFORM BREAK PROCESSING が実行されると、Natural によって、ブレイクが起こったかどうか、つまり、指定したコントロールフィールドの値が変わったかどうかがチェックされます。値が変わっている場合、指定したステートメントが実行されます。

PERFORM BREAK PROCESSING では、ブレイクが起こったかどうかをチェックする*前*に、システム関数が評価されます。

以下の図は、ユーザー開始のブレイク処理のフローロジックを示しています。



PERFORM BREAK PROCESSING ステートメントの例

```
** Example 'PERFBX01': PERFORM BREAK PROCESSING (with BREAK option
**                      in IF statement)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 DEPT
  2 SALARY (1:1)
*
1 #CNTL    (N2)
END-DEFINE
*
LIMIT 7
READ MYVIEW BY DEPT
  AT BREAK OF DEPT              /* <- automatic break processing
    SKIP 1
    WRITE 'SUMMARY FOR ALL SALARIES         '
          'SUM:'   SUM(SALARY(1))
          'TOTAL:' TOTAL(SALARY(1))
    ADD 1 TO #CNTL
  END-BREAK
  /*
  IF SALARY (1) GREATER THAN 100000 OR BREAK #CNTL
    PERFORM BREAK PROCESSING    /* <- user-initiated break processing
    AT BREAK OF #CNTL
      WRITE 'SUMMARY FOR SALARY GREATER 100000'
            'SUM:'   SUM(SALARY(1))
            'TOTAL:' TOTAL(SALARY(1))
    END-BREAK
  END-IF
  /*
  IF SALARY (1) GREATER THAN 150000 OR BREAK #CNTL
    PERFORM BREAK PROCESSING    /* <- user-initiated break processing
    AT BREAK OF #CNTL
      WRITE 'SUMMARY FOR SALARY GREATER 150000'
            'SUM:'   SUM(SALARY(1))
            'TOTAL:' TOTAL(SALARY(1))
    END-BREAK
  END-IF
  DISPLAY NAME DEPT SALARY(1)
END-READ
END
```

プログラム PERFBX01 の出力：

```
Page      1                                                 04-12-14  14:13:35

        NAME          DEPARTMENT   ANNUAL
                         CODE      SALARY
-------------------- ---------- ----------

JENSEN               ADMA01         180000
PETERSEN             ADMA01         105000
MORTENSEN            ADMA01         320000
MADSEN               ADMA01         149000
BUHL                 ADMA01         642000

SUMMARY FOR ALL SALARIES          SUM:    1396000 TOTAL:    1396000
SUMMARY FOR SALARY GREATER 100000 SUM:    1396000 TOTAL:    1396000
SUMMARY FOR SALARY GREATER 150000 SUM:    1142000 TOTAL:    1142000
HERMANSEN            ADMA02         391500
PLOUG                ADMA02         162900

SUMMARY FOR ALL SALARIES          SUM:     554400 TOTAL:    1950400
SUMMARY FOR SALARY GREATER 100000 SUM:     554400 TOTAL:    1950400
SUMMARY FOR SALARY GREATER 150000 SUM:     554400 TOTAL:    1696400
```

45 データ計算

この章では、データ計算に使用する、以下の算術演算ステートメントについて説明します。

- COMPUTE
- ADD
- SUBTRACT
- MULTIPLY
- DIVIDE

また、1つのオペランドの値を1つ以上のフィールドに転送するために使用する、以下のステートメントについて説明します。

- MOVE
- COMPRESS

重要: 最適な処理を行うために、算術演算ステートメントに使用するユーザー定義変数は、フォーマットP（パック型数値）で定義してください。

COMPUTE ステートメント

COMPUTE ステートメントは、算術演算を実行するために使用します。 以下の結合演算子を使用できます。

**	累乗
*	乗算
/	除算
+	加算
-	減算
()	カッコは、論理グループを示すために使用します。

例 **1**：

```
COMPUTE LEAVE-DUE = LEAVE-DUE * 1.1
```

この例では、フィールド LEAVE-DUE の値を 1.1 で乗算して、その結果をフィールド LEAVE-DUE に設定しています。

例 **2**：

```
COMPUTE #A = SQRT (#B)
```

この例では、フィールド #B の値の平方根を算出して、その結果をフィールド #A に割り当てています。

SQRT は、以下の算術演算ステートメントでサポートされている算術関数です。

- COMPUTE
- ADD
- SUBTRACT
- MULTIPLY
- DIVIDE

算術関数の概要については、後述の「算術関数」を参照してください。

例 **3**：

```
COMPUTE #INCOME = BONUS (1,1) + SALARY (1)
```

この例では、当年の最初のボーナスと現在の給与を加算して、その結果をフィールド #INCOME に割り当てています。

MOVE および COMPUTE ステートメント

ステートメント MOVE および COMPUTE は、オペランドの値を 1 つ以上のフィールドに転送するために使用できます。 オペランドには、テキスト項目や数値などの定数、データベースフィールド、ユーザー定義変数、システム変数、または場合によってはシステム関数を使用できます。

これらの 2 つのステートメントの違いは、以下の例のように、MOVE ステートメントでは移動元の値を左側に指定しますが、COMPUTE ステートメントでは割り当て元の値を右側に指定します。

例：

```
MOVE NAME TO #LAST-NAME
COMPUTE #LAST-NAME = NAME
```

ADD、SUBTRACT、MULTIPLY、および DIVIDE ステートメント

ADD、SUBTRACT、MULTIPLY、および DIVIDE ステートメントは、算術演算を実行するために使用します。

例：

```
ADD +5 -2 -1 GIVING #A
SUBTRACT 6 FROM 11 GIVING #B
MULTIPLY 3 BY 4 GIVING #C
DIVIDE 3 INTO #D GIVING #E
```

これらの 4 つのステートメントはすべて ROUNDED オプションを持っています。このオプションは、演算の結果を四捨五入する場合に使用します。

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。

これらのステートメントの詳細については、『ステートメント』ドキュメントを参照してください。

MOVE、SUBTRACT、および COMPUTE ステートメントの例

以下のプログラムは、算術演算ステートメントでのユーザー定義変数の使用方法を示しています。 3 人の従業員の年齢と給与を計算し、その結果を出力します。

```
** Example 'COMPUX01': COMPUTE
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 BIRTH
  2 JOB-TITLE
  2 SALARY          (1:1)
  2 BONUS           (1:1,1:1)
*
1 #DATE             (N8)
1 REDEFINE #DATE
  2 #YEAR           (N4)
  2 #MONTH          (N2)
  2 #DAY            (N2)
1 #BIRTH-YEAR       (A4)
1 REDEFINE #BIRTH-YEAR
  2 #BIRTH-YEAR-N   (N4)
```

```
1 #AGE              (N3)
1 #INCOME           (P9)
END-DEFINE
*
MOVE *DATN TO #DATE
*
READ (3) MYVIEW BY NAME STARTING FROM 'JONES'
  MOVE EDITED BIRTH (EM=YYYY) TO #BIRTH-YEAR
  SUBTRACT #BIRTH-YEAR-N FROM #YEAR GIVING #AGE
  /*
  COMPUTE #INCOME = BONUS (1:1,1:1) + SALARY (1:1)
  /*
  DISPLAY NAME 'POSITION' JOB-TITLE #AGE #INCOME
END-READ
END
```

プログラム COMPUX01 の出力：

```
Page      1                                                 04-11-11  14:15:54

        NAME                  POSITION           #AGE  #INCOME
-------------------- ------------------------- ---- ----------

JONES                MANAGER                     63      55000
JONES                DIRECTOR                    58      50000
JONES                PROGRAMMER                  48      31000
```

COMPRESS ステートメント

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

数値フィールドの先行ゼロおよび英数字フィールドの末尾の空白は、フィールド値を受け取りフィールドに転送する前に削除されます。

デフォルトでは、転送される値はそれぞれ1つの空白で区切られ、受け取りフィールドに格納されます。 その他に可能な分割方法については、『ステートメント』ドキュメントに記載されている、COMPRESS ステートメントオプション LEAVING NO SPACE の説明を参照してください。

例：

```
COMPRESS 'NAME:' FIRST-NAME #LAST-NAME INTO #FULLNAME
```

この例では、COMPRESS ステートメントを使用して、テキスト定数（'NAME:'）、データベースフィールド（FIRST-NAME）、およびユーザー定義変数（#LAST-NAME）を、1 つのユーザー定義変数（#FULLNAME）に結合しています。

COMPRESS ステートメントの詳細については、『ステートメント』ドキュメントに記載されている、COMPRESS ステートメントの説明を参照してください。

COMPRESS および MOVE ステートメントの例

以下のプログラムは、ステートメント MOVE および COMPRESS の使用方法を示しています。

```
** Example 'COMPRX01': COMPRESS
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 MIDDLE-I
*
1 #LAST-NAME (A15)
1 #FULL-NAME (A30)
END-DEFINE
*
READ (3) MYVIEW BY NAME STARTING FROM 'JONES'
  MOVE NAME TO #LAST-NAME
  /*
  COMPRESS 'NAME:' FIRST-NAME MIDDLE-I #LAST-NAME INTO #FULL-NAME
  /*
  DISPLAY #FULL-NAME (UC==) FIRST-NAME 'I' MIDDLE-I (AL=1) NAME
END-READ
END
```

プログラム COMPRX01 の出力：

結合されたフィールドの出力形式に注意してください。

```
Page      1                                                  04-11-11  14:15:54

          #FULL-NAME                FIRST-NAME      I          NAME
============================== ------------------- - --------------------

NAME: VIRGINIA J JONES         VIRGINIA            J JONES
NAME: MARSHA JONES             MARSHA                JONES
NAME: ROBERT B JONES           ROBERT              B JONES
```

複数行表示では、COMPRESS ステートメントを使用して、ユーザー定義変数にフィールド／テキストを結合すると便利です。

COMPRESS ステートメントの例

以下のプログラムでは、#FULL-SALARY、#FULL-NAME、および #FULL-CITY の 3 つのユーザー定義変数を使用しています。例えば、#FULL-SALARY には、テキスト 'SALARY:' と、データベースフィールドの SALARY および CURR-CODE が格納されます。その後、WRITE ステートメントでは、結合した変数のみを参照しています。

```
** Example 'COMPRX02': COMPRESS
************************************************************************
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 SALARY       (1:1)
  2 CURR-CODE    (1:1)
  2 CITY
  2 ADDRESS-LINE (1:1)
  2 ZIP
*
1 #FULL-SALARY   (A25)
1 #FULL-NAME     (A25)
1 #FULL-CITY     (A25)
END-DEFINE
*
READ (3) VIEWEMP BY CITY STARTING FROM 'NEW YORK'
  COMPRESS 'SALARY:' CURR-CODE(1) SALARY(1) INTO #FULL-SALARY
  COMPRESS FIRST-NAME NAME                  INTO #FULL-NAME
  COMPRESS ZIP CITY                         INTO #FULL-CITY
  /*
  DISPLAY 'NAME AND ADDRESS' NAME (EM=X^X^X^X^X^X^X^X^X^X^X^X)
  WRITE 1/5  #FULL-NAME
        1/37 #FULL-SALARY
        2/5  ADDRESS-LINE (1)
        3/5  #FULL-CITY
```

```
  SKIP 1
END-READ
END
```

プログラム COMPRX02 の出力：

```
Page      1                                                  04-11-11  14:15:54

   NAME AND ADDRESS
----------------------

R U B I N
    SYLVIA RUBIN                    SALARY: USD 17000
    2003 SARAZEN PLACE
    10036 NEW YORK

W A L L A C E
    MARY WALLACE                    SALARY: USD 38000
    12248 LAUREL GLADE C
    10036 NEW YORK

K E L L O G G
    HENRIETTA KELLOGG               SALARY: USD 52000
    1001 JEFF RYAN DR.
    19711 NEWARK
```

算術関数

以下の Natural 算術関数は、算術演算ステートメント（ADD、COMPUTE、DIVIDE、SUBTRACT、MULTIPLY）でサポートされています。

算術の内容	Natural システム関数
field の絶対値。	ABS(*field*)
field アークタンジェント。	ATN(*field*)
field のコサイン。	COS(*field*)
field の指数。	EXP(*field*)
field の小数部分。	FRAC(*field*)
field の整数部分。	INT(*field*)
field の自然対数。	LOG(*field*)
field.の符号。	SGN(*field*)
field のサイン（正弦）。	SIN(*field*)
field の平方根。	SQRT(*field*)

算術の内容	Natural システム関数
field のタンジェント。	TAN(*field*)
英数字 *field* から抽出された数値。	VAL(*field*)

各算術関数の詳細については、『システム*関数*』ドキュメントも参照してください。

COMPUTE、MOVE、および COMPRESS ステートメントの他の例

次の例のプログラムを参照してください。

- ***WRITEX11 - WRITE（nX、n/n および COMPRESS を使用）***
- ***IFX03 - IF ステートメント***
- ***COMPRX03 - COMPRESS（パラメータ LC および TC を使用）***

46 システム変数とシステム関数

この章では、Natural システム変数と Natural システム関数の用途、および Natural プログラムでの使用方法を説明します。

システム変数

以下では次のトピックについて説明します。

用途

システム変数を使用して、システム情報を表示します。 これらは、Naturalプログラム内のどこからでも参照できます。

Natural システム変数は、常に変わる情報、例えば以下のような、現在の Natural セッションに関する情報を提供します。

- 現在のライブラリ
- ユーザーおよび端末 ID
- ループ処理の現在のステータス
- レポート処理の現在のステータス
- 現在の日付と時刻

システム変数の一般的な使用方法については、後述の「*システム変数およびシステム関数の使用例*」、およびライブラリ **SYSEXPG** に含まれている例を参照してください。

システム変数に格納される情報は、Naturalプログラムで適切なシステム変数を指定することにより、使用できます。 例えば、日付と時刻のシステム変数は、`DISPLAY`、`WRITE`、`PRINT`、`MOVE`、または `COMPUTE` の各ステートメントで指定できます。

システム変数の特徴

システム変数の名前はすべて、アスタリスク（*）で始まります。

フォーマット／長さ

システム変数のフォーマット／長さの情報については、『システム変数』ドキュメントを参照してください。 次の省略形が使用されます。

フォーマット	
A	英数字
B	バイナリ
D	日付
I	整数
L	論理
N	数値（アンパック）
P	パック型数値
T	時刻

内容変更の可否

それぞれの説明に記載されているこの項目は、Naturalプログラム内で別の値をシステム変数に割り当てることができるかどうか、つまり、Naturalによって生成された内容を上書きできるかどうかを示します。

機能別のシステム変数

Natural システム変数は、以下のように分類されます。

- アプリケーション関連システム変数
- 日時システム変数
- 入出力関連システム変数
- Natural 環境関連システム変数
- システム環境関連システム変数
- XML 関連システム変数

すべて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。

システム関数

Naturalシステム関数は、レコードを処理した後（ただしブレイク処理の発生前）にデータに適用できる統計関数および算術関数によって構成されます。

システム関数は、AT END OF PAGE、AT END OF DATA、またはAT BREAKの各ステートメントとともに使用されるDISPLAY、WRITE、PRINT、MOVE、またはCOMPUTEの各ステートメントに指定できます。

AT END OF PAGEステートメントの場合、対応するDISPLAYステートメントでGIVE SYSTEM FUNCTIONS節を使用する必要があります（以下の例を参照）。

システム関数は、以下の機能に分類されます。

- 処理ループで使用する Natural システム関数
- 算術関数
- その他の関数

有効なすべての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。

『*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*」も参照してください。

システム関数の一般的な使用方法については、以下のプログラム例およびライブラリ**SYSEXPG**に格納されている例を参照してください。

システム変数およびシステム関数の使用例

以下のプログラム例は、システム変数およびシステム関数の使用方法を示しています。

```
** Example 'SYSVAX01': System variables and system functions
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 JOB-TITLE
  2 INCOME    (1:1)
    3 CURR-CODE
    3 SALARY
    3 BONUS   (1:1)
END-DEFINE
```

```
*
WRITE TITLE LEFT JUSTIFIED 'EMPLOYEE SALARY REPORT AS OF' *DAT4E /
*
READ (3) MYVIEW BY CITY STARTING FROM 'E'
  DISPLAY GIVE SYSTEM FUNCTIONS
          NAME (AL=15) JOB-TITLE (AL=15) INCOME (1:1)
  AT START OF DATA
    WRITE 'REPORT CREATED AT:' *TIME 'HOURS' /
  END-START
  AT END OF DATA
    WRITE / 'LAST PERSON SELECTED:' OLD (NAME) /
  END-ENDDATA
END-READ
*
AT END OF PAGE
  WRITE 'AVERAGE SALARY:' AVER (SALARY(1))
END-ENDPAGE
END
```

説明：

- システム変数 *DATE は、WRITE TITLE ステートメントで出力されています。
- システム変数 *TIME は、AT START OF DATA ステートメントで出力されています。
- システム関数 OLD は、AT END OF DATA ステートメントで使用されています。
- システム関数 AVER は、AT END OF PAGE ステートメントで使用されています。

プログラム SYSVAX01 の出力：

システム変数およびシステム関数がどのように表示されているかに注意してください。

```
EMPLOYEE SALARY REPORT AS OF 11/11/2004

     NAME           CURRENT                  INCOME
                    POSITION
                                   CURRENCY   ANNUAL     BONUS
                                     CODE     SALARY
--------------- --------------- -------- ---------- ----------

REPORT CREATED AT: 14:15:55.0 HOURS

DUYVERMAN       PROGRAMMER      USD           34000          0
PRATT           SALES PERSON    USD           38000       9000
MARKUSH         TRAINEE         USD           22000          0

LAST PERSON SELECTED: MARKUSH

AVERAGE SALARY:      31333
```

システム変数の他の例

次の例のプログラムを参照してください。

- *EDITMX05 - 編集マスク（日付および時刻システム変数の EM）*
- *READX04 - READ（FIND およびシステム変数 *NUMBER と *COUNTER を使用）*
- *WTITLX01 - WRITE TITLE（PAGE-NUMBER を使用）*

システム関数の他の例

次の例のプログラムを参照してください。

- *ATBREX06 - AT BREAK OF（NMIN、NAVER、NCOUNT を MIN、AVER、COUNT と比較）*
- *ATENPX01 - AT END OF PAGE（DISPLAY の GIVE SYSTEM FUNCTIONS で有効なシステム関数を使用）*

47 スタック

Natural スタックは、Natural コマンド、ユーザー定義コマンド、および INPUT ステートメントで使用する入力データを格納できる、一種の 「中間ストレージ」 です。

Natural スタックの使用

スタックには、一連のログオンコマンドのように、頻繁に連続して実行する一連の機能を格納できます。

スタックに格納されたデータ／コマンドは、その一番上に 「スタック」 されます。 格納するデータ／コマンドをスタックの一番上と一番下のどちらに置くかは指定できます。 スタック内のデータ／コマンドは、スタックされている順番（スタックの一番上から）でのみ処理できます。

プログラムでは、システム変数 *DATA を参照することにより、スタックの内容を確認できます。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。

スタック処理

スタックに格納されているコマンド／データの処理は、実行する機能によって異なります。

コマンドを実行しようとしている場合、つまり、NEXT プロンプトを表示しようとしている場合、Natural はまず、スタックの一番上がコマンドかどうかをチェックします。 スタックの一番上がコマンドの場合、NEXT プロンプトは抑制されます。そして、コマンドが読み込まれ、スタックから削除されます。その後、NEXT プロンプトに応答して手動で入力されたかのように、コマンドが実行されます。

入力フィールドが含まれている INPUT ステートメントを実行しようとしている場合、Natural はまず、スタックの一番上に何らかの入力データがあるかどうかをチェックします。 入力データがある場合、そのデータが INPUT ステートメントにデリミタモードで渡されます。スタックから読み込まれるデータは、INPUT ステートメントの変数と互換性のあるフォーマットである必要があります。その後、データはスタックから削除されます。 INPUT ステートメントの説明の「*Natural スタックデータの処理*」も参照してください。

スタックデータを使用して実行した INPUT ステートメントを REINPUT ステートメント経由で再実行すると、INPUT ステートメントの画面が再表示され、最初に実行したときと同じスタックデータが表示されます。 REINPUT ステートメントでは、スタックから他のデータは読み込まれません。

Natural プログラムが正常に終了するときは、コマンドがスタックの一番上に来るか、またはスタックがクリアされるまで、スタックは一番上からフラッシュされます。Natural プログラムが端末コマンド %% またはエラーで終了した場合は、スタックが完全にクリアされます。

スタックへのデータの格納

以下の方法を使用して、データ／コマンドをスタックに格納します。

- STACK パラメータ
- STACK ステートメント
- FETCH および RUN ステートメント

STACK パラメータ

Natural プロファイルパラメータ STACK は、データ／コマンドをスタックに格納するために使用できます。 STACK パラメータ（『パラメータリファレンス』を参照）は、Natural のインストール時に Natural 管理者が Natural パラメータモジュールに指定するか、またはダイナミックパラメータとして Natural の起動時に指定します。

データ／コマンドを STACK パラメータ経由でスタックに格納する場合、複数のコマンドはセミコロン（;）で分割する必要があります。 コマンドを一連のデータまたはコマンド要素に格納して渡す場合、コマンドの前にセミコロンを付ける必要があります。

複数の INPUT ステートメントに対するデータは、コロン（:）で分割する必要があります。 単独の INPUT ステートメントで読み込むデータには、コロンを前に付ける必要があります。パラメータを必要とするコマンドをスタックする場合、コマンドとパラメータの間にはコロンを置きません。

セミコロンとコロンはでセパレータ文字として解釈されるため、入力データ自体にセミコロンとコロンを使用することはできません。

STACK ステートメント

プログラム内で STACK ステートメントを使用すると、データ／コマンドをスタックに格納できます。 1 つの STACK ステートメントで指定したデータ要素は、1 つの INPUT ステートメントで使用します。これは、複数の INPUT ステートメントに対するデータをスタックに格納するには、複数の STACK ステートメントを使用する必要があることを意味します。

スタックには、フォーマットされていないデータもフォーマットされているデータも格納できます。

- フォーマットされていないデータがスタックから読み込まれると、データ文字列はデリミタモードで解釈され、セッションパラメータ IA（INPUT 割り当て文字）および ID（INPUT 区切り文字）で指定されている文字が、割り当てとデータ分割のキーワードに対する制御文字として処理されます。
- フォーマットされているデータがスタックに格納されている場合、フィールドの個々の内容が分割され、対応する INPUT ステートメントの 1 つの入力フィールドに渡されます。 スタックに格納す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、フォーマットしてスタックに格納する必要があります。

STACK ステートメントの詳細については、『ステートメント』ドキュメントを参照してください。

FETCH および RUN ステートメント

呼び出すプログラムに渡すパラメータを持つ FETCH または RUN ステートメントを実行すると、これらのパラメータはスタックの一番上に格納されます。

スタックの最初のエントリの削除

Natural 端末コマンド %.P は、Natural スタックの一番上のエントリを削除します。

スタックのクリア

スタックの内容は、RELEASE ステートメントで削除できます。 RELEASE ステートメントの詳細については、『ステートメント』ドキュメントを参照してください。

注意: Natural プログラムが端末コマンド %%またはエラーで終了した場合は、スタックが完全にクリアされます。

48 日付情報の処理

この章では、Naturalアプリケーションでの日付情報の処理について、さまざまな面から説明します。

日付フィールドの編集マスクおよび日付システム変数

日付フィールドの値を特定の表現で出力する場合は、通常、フィールドに編集マスクを指定します。 編集マスクを使用して、1 文字ずつどのように出力するかを指定します。

現在の日付を特定の表現で使用する場合、日付フィールドを定義して編集マスクを指定する必要はありません。代わりに、*日付システム変数*を使用します。Naturalには、さまざまな日付システム変数が用意されています。その中には、表現の異なる現在日付も含まれています。 これらの表現には 2 桁の年コンポーネントを持つものもあれば、4 桁の年コンポーネントを持つものもあります。

すべての日付システム変数のリストと詳細については、『システム変数』ドキュメントを参照してください。

デフォルトの日付編集マスク - DTFORM パラメータ

DTFORM プロファイルパラメータによって、Natural レポートのデフォルトタイトルの日付部分、日付定数、および日付入力に使用されるデフォルトのフォーマットが決まります。

この日付フォーマットを使用して、日付の年、月、日の各コンポーネントの順序、およびこれらのコンポーネント間で使用するデリミタ文字を指定します。

有効な DTFORM の設定は、以下のとおりです。

設定	日付フォーマット*	例
DTFORM=I	*yyyy-mm-dd*	2005-12-31
DTFORM=G	*dd.mm.yyyy*	31.12.2005
DTFORM=E	*dd/mm/yyyy*	31/12/2005
DTFORM=U	*mm/dd/yyyy*	12/31/2005

* *dd* = 日、*mm* = 月、*yyyy* = 年。

DTFORM パラメータは、Natural パラメータモジュール／ファイル内で設定できます。また、Naturalを起動するときにダイナミックに設定することもできます。 デフォルトでは、DTFORM=I が適用されます。

英数字表現の日付フォーマット - DF パラメータ

編集マスクが指定されている場合、フィールド値の表現は編集マスクによって決まります。 編集マスクが指定されていない場合、フィールド値の表現は、セッションパラメータ DF とプロファイルパラメータ DTFORM の組み合せによって決まります。

DF パラメータでは、以下の日付表現の 1 つを選択できます。

DF=S	2 桁の年コンポーネントとデリミタを使用する 8 バイトの表記です（*yy-mm-dd*）。
DF=I	デリミタを使用しない、4 桁の年コンポーネントの 8 バイトの表記です（*yyyymmdd*）。
DF=L	4 桁の年コンポーネントとデリミタを使用する 10 バイトの表記です（*yyyy-mm-dd*）。

各表現に対する、年、月、日の各コンポーネントの順序、および使用するデリミタ文字は、DTFORM パラメータによって決まります。

デフォルトでは、DF=S が適用されます（INPUT ステートメントを除く。下記参照）。

セッションパラメータ DF は、コンパイル時に評価されます。

このパラメータは、以下のステートメントで指定できます。

- FORMAT
- INPUT、DISPLAY、WRITE、および PRINT のステートメントレベルおよび要素（フィールド）レベル
- MOVE、COMPRESS、STACK、RUN、および FETCH の要素（フィールド）レベル

上記のステートメントのいずれかで指定すると、DF パラメータは以下のように適用されます。

ステートメント	DF パラメータの効果
DISPLAY、WRITE、PRINT	これらのステートメントの1つを使用して日付変数の値を出力すると、出力する前に値が英数字表現に変換されます。 DF パラメータにより、使用される表現が決まります。
MOVE、COMPRESS	MOVE または COMPRESS の各ステートメントを使用して日付変数の値を英数字フィールドに転送すると、転送する前に値が英数字表現に変換されます。 DF パラメータにより、使用される表現が決まります。
STACK、RUN、FETCH	日付変数の値をスタックに格納すると、スタックに格納する前に値が英数字表現に変換されます。 DF パラメータにより、使用される表現が決まります。 FETCH ステートメントまたは RUN ステートメントのパラメータとして日付変数を指定する場合も同様です（これらのパラメータもスタック経由で渡されるため）。

ステートメント	DF パラメータの効果
INPUT	INPUT ステートメントでデータ変数を使用する場合、DF パラメータによって、値をどのようにフィールドに入力する必要があるかが決まります。 ただし、DF パラメータを*指定せずに*日付変数を INPUT ステートメントで使用すると、デリミタ付きの2桁の年コンポーネント、またはデリミタなしの4桁の年コンポーネントのいずれかで日付を入力できます。 この場合も、年、月、日の各コンポーネントの順序、および使用するデリミタ文字は、DTFORM パラメータによって決まります。

注意: DF=S で提供される年情報は 2 桁のみです。これは、日付の値に世紀が含まれていると、変換時にこの情報が失われることを意味します。 世紀の情報を保持するには、DF=I または DF=L を設定します。

WRITE ステートメントでの DF パラメータの例

これらの例では、DTFORM=G が適用されると仮定します。

```
/* DF=S (default)
WRITE *DATX   /* Output has this format: dd.mm.yy
END
```

```
FORMAT DF=I
WRITE *DATX   /* Output has this format: ddmmyyyy
END
```

```
FORMAT DF=L
WRITE *DATX   /* Output has this format: dd.mm.yyyy
END
```

MOVE ステートメントでの DF パラメータの例

この例では、DTFORM=E が適用されると仮定します。

```
DEFINE DATA LOCAL
  1 #DATE  (D) INIT <D'31/12/2005'>
  1 #ALPHA (A10)
END-DEFINE
...
MOVE #DATE        TO #ALPHA /* Result: #ALPHA contains 31/12/05
MOVE #DATE (DF=I) TO #ALPHA /* Result: #ALPHA contains 31122005
MOVE #DATE (DF=L) TO #ALPHA /* Result: #ALPHA contains 31/12/2005
...
```

STACK ステートメントでの DF パラメータの例

この例では、DTFORM=I が適用されると仮定します。

```
DEFINE DATA LOCAL
  1 #DATE  (D) INIT <D'2005-12-31'>
  1 #ALPHA1(A10)
  1 #ALPHA2(A10)
  1 #ALPHA3(A10)
END-DEFINE
...
STACK TOP DATA #DATE (DF=S) #DATE (DF=I) #DATE (DF=L)
...
INPUT #ALPHA1 #ALPHA2 #ALPHA3
...
/* Result: #ALPHA1 contains 05-12-31
/*         #ALPHA2 contains 20051231
/*         #ALPHA3 contains 2005-12-31
...
```

INPUT ステートメントでの DF パラメータの例

この例では、DTFORM=I が適用されると仮定します。

```
DEFINE DATA LOCAL
  1 #DATE1 (D)
  1 #DATE2 (D)
  1 #DATE3 (D)
  1 #DATE4 (D)
END-DEFINE
...
INPUT #DATE1 (DF=S) /* Input must have this format: yy-mm-dd
      #DATE2 (DF=I) /* Input must have this format: yyyymmdd
      #DATE3 (DF=L) /* Input must have this format: yyyy-mm-dd
      #DATE4        /* Input must have this format: yy-mm-dd or yyyymmdd
...
```

出力用の日付フォーマット - DFOUT パラメータ

セッション／プロファイルパラメータ DFOUT は、編集マスクが指定されておらず、DF パラメータも適用されていない、INPUT、DISPLAY、WRITE、および PRINT の各ステートメントの日付フィールドにのみ適用されます。

INPUT、DISPLAY、PRINT、および WRITE の各ステートメントによって表示され、編集マスクが指定されておらず、DF パラメータも適用されてない日付フィールドの場合は、プロファイル／セッションパラメータ DFOUT によって、表示するフィールド値のフォーマットが決まります。

有効な DFOUT の設定は、以下のとおりです。

DFOUT=S	日付変数は、2 桁の年コンポーネント、およびパラメータ DTFORM で指定されているデリミタを使用して表示されます（*yy-mm-dd*）。
DFOUT=I	日付変数は、デリミタなしの 4 桁の年コンポーネントで表示されます（*yyyymmdd*）。

デフォルトでは、DFOUT=S が適用されます。 どちらの DFOUT 設定でも、日付値の年、月、日の各コンポーネントの順序は、DTFORM パラメータによって決まります。

どちらの日付値表現でも 8 バイトのフィールドに収まるため、日付フィールドの長さは DFOUT 設定の影響を受けません。

DFOUT パラメータは、Natural パラメータモジュール／ファイル内で、または Natural を起動するときにダイナミックに、あるいはシステムコマンド GLOBALS を使用して設定できます。 これは、ランタイム時に評価されます。

例：

この例では、DTFORM=I が適用されると仮定します。

```
DEFINE DATA LOCAL
1 #DATE  (D) INIT <D'2005-12-31'>
END-DEFINE
...
WRITE #DATE        /* Output if DFOUT=S is set ...: 05-12-31
                   /* Output if DFOUT=I is set ...: 20051231
WRITE #DATE (DF=L) /* Output (regardless of DFOUT): 2005-12-31
...
```

スタック用の日付フォーマット - DFSTACK パラメータ

セッション／プロファイルパラメータ DFSTACK は、DF パラメータが指定されていない、STACK、FETCH、および RUN の各ステートメントで使用されている日付フィールドにのみ適用されます。

DFSTACK パラメータによって、STACK、RUN、FETCH の各ステートメント経由でスタックに格納される日付変数の値のフォーマットが決まります。

有効な DFSTACK の設定は、以下のとおりです。

DFSTACK=S	日付変数は、2 桁の年コンポーネント、およびプロファイルパラメータ DTFORM で指定されているデリミタを使用してスタックに格納されます（*yy-mm-dd*）。
DFSTACK=C	DFSTACK=S と同じです。 ただし、世紀の変更はランタイム時にインターセプトされます。
DFSTACK=I	日付変数は、デリミタなしの 4 桁の年コンポーネントでスタックに格納されます（*yyyymmdd*）。

デフォルトでは、DFSTACK=S が適用されます。 DFSTACK=S は、世紀の情報なし（つまり、失われます）で日付値がスタックに格納されることを意味します。 その後、値をスタックから読み込んで別の日付変数に格納すると、世紀情報は現在のものであるとみなされるか、または YSLW パラメータ（下記参照）の設定によって決まります。 この操作によって、元の日付値と異なる世紀が設定される可能性があります。ただし、この場合、Natural はエラーを発行しません。

世紀の情報なしで日付値がスタックに格納されるという点では、DFSTACK=C は DFSTACK=S と同じ動作をします。 ただし、（YSLW パラメータ、または元の世紀が現在の世紀でなかったために）スタックから読み込んだ値の世紀情報が元の日付値と異なった場合、Naturalはランタイムエラーを発行します。

注意: このランタイムエラーは、値をスタックに格納する時点ですでに発行されています。

DFSTACK=I を使用すると、世紀の情報を失わずに日付値を8バイトの長さでスタックに格納できます。

DFSTACK パラメータは、Natural パラメータモジュール／ファイル内で、または Natural を起動するときにダイナミックに、あるいはシステムコマンド GLOBALS を使用して設定できます。 これは、ランタイム時に評価されます。

例：

以下の例では、DTFORM=I および YSLW=0 が適用されているものとします。

```
DEFINE DATA LOCAL
  1 #DATE  (D) INIT <D'2005-12-31'>
  1 #ALPHA1(A8)
  1 #ALPHA2(A10)
END-DEFINE
...
STACK TOP DATA #DATE #DATE (DF=L)
...
INPUT #ALPHA1 #ALPHA2
...
/* Result if DFSTACK=S or =C is set: #ALPHA1 contains 05-12-31
/* Result if DFSTACK=I is set .....: #ALPHA1 contains 20051231
/* Result (regardless of DFSTACK) .: #ALPHA2 contains 2005-12-31
...
```

年スライディングウィンドウ - YSLW パラメータ

プロファイルパラメータ YSLW を使用すると、2 桁の年の値に対する世紀を指定できます。

YSLW パラメータは、Natural パラメータモジュール／ファイル内で設定できます。また、Natural を起動するときにダイナミックに設定することもできます。2桁の年コンポーネントを持つ英数字の日付値を日付変数に移すと、このパラメータはランタイム時に評価されます。 これは、以下のデータ値に適用されます。

- 算術関数 VAL(*field*) で使用するデータ値
- 論理条件の IS(D) オプションで使用するデータ値
- 入力データとしてスタックから読み込まれたデータ値
- 入力データとして入力フィールドに入力されたデータ値

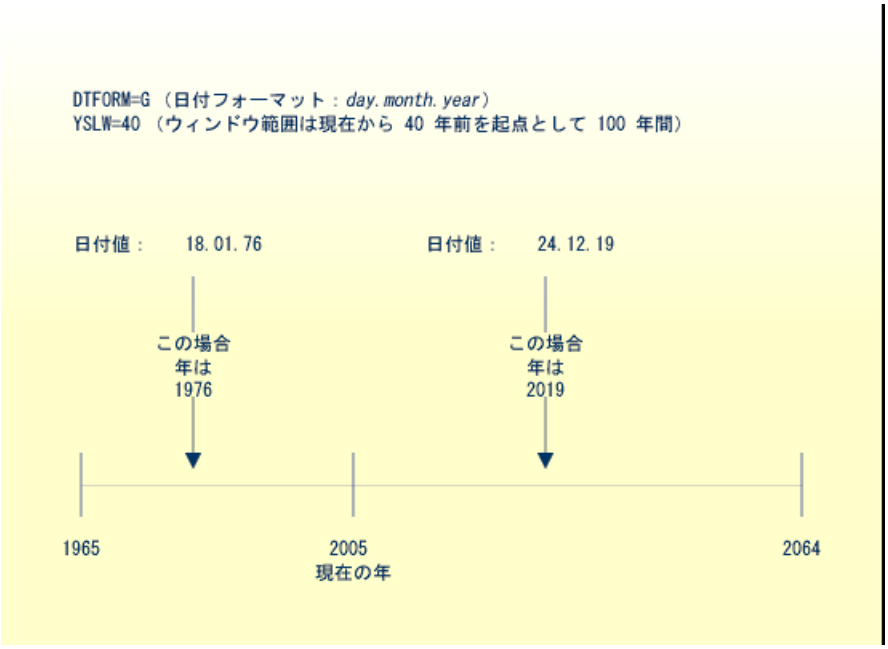
YSLW パラメータによって、「年スライディングウィンドウ」 でカバーする年の範囲を特定します。 スラインディングウィンドウのメカニズムは、2 桁の年の日付が 100 年の 「ウィンドウ」 内にあるものと仮定します。 この 100 年の範囲で、すべての 2 桁の年の値を特定の世紀に一意に関連付けることができます。

YSLW パラメータを使用して、その 100 年の範囲を何年前から開始するかを指定します。現在の年から YSLW の値を引くことにより、ウィンドウ範囲の最初の年が決定します。

YSLW パラメータの有効な値は、0～99 です。 デフォルト値は YSLW=0 です。これは、スライディングウィンドウメカニズムを使用しないことを意味します。つまり、2桁の年を持つ日付は、現在の世紀にあるとみなされます。

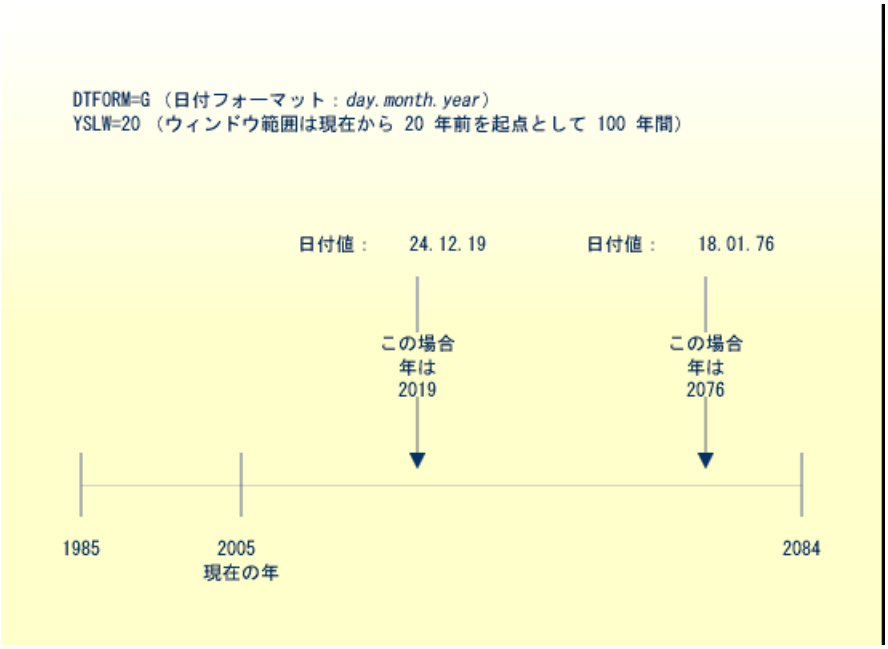
例 **1**：

現在の年が 2005 年の場合に YSLW=40 を指定すると、スラインディングウィンドウは 1965～2064 年をカバーします。 65～99 の 2 桁の年の値 *nn* は、19*nn* と解釈されます。一方、00～64 の 2 桁の年の値 *nn* は、20*nn* と解釈されます。



例 **2**：

現在の年が 2005 年の場合に YSLW=20 を指定すると、スラインディングウィンドウは 1985～2084 年をカバーします。 85～99 の 2 桁の年の値 *nn* は、19*nn* と解釈されます。一方、00～84 の 2 桁の年の値 *nn* は、20*nn* と解釈されます。



DFSTACK と YSLW の組み合わせ

以下の例で、パラメータ DFSTACK と YSLW をさまざまに組み合わせて使用することによる効果を説明します。

注意: これらのすべての例において、DTFORM=I が適用されているものとします。

例 **1**：

以下の例では、現在の年は 2005 年で、パラメータの設定には DFSTACK=S（デフォルト値）および YSLW=20 が適用されているものとします。

```
DEFINE DATA LOCAL
  1 #DATE1 (D) INIT <D'1956-12-31'>
  1 #DATE2 (D)
END-DEFINE
...
STACK TOP DATA #DATE1 /* century information is lost (year 56 is stacked)
...
INPUT #DATE2          /* year sliding window determines 56 to be 2056
...
/* Result: #DATE2 contains 2056-12-31
           even if #DATE1 is set to <D'2156-12-31'>
```

この場合、年スライディングウィンドウは適切に設定されないため、世紀の情報は（意図しない値に）変更されます。

例 **2**：

以下の例では、現在の年は 2005 年で、パラメータの設定には DFSTACK=S（デフォルト値）および YSLW=60 が適用されているものとします。

```
DEFINE DATA LOCAL
  1 #DATE1 (D) INIT <D'1956-12-31'>
  1 #DATE2 (D)
END-DEFINE
...
STACK TOP DATA #DATE1 /* century information is lost (year 56 is stacked)
...
INPUT #DATE2          /* year sliding window determines 56 to be 1956
...
/* Result: #DATE2 contains 1956-12-31
           even if #DATE1 is set to <D'2056-12-31'>
```

この場合、年スライディングウィンドウは適切に設定されるため、世紀の情報は正しく復元されます。

例 **3**：

以下の例では、現在の年は 2005 年で、パラメータの設定には DFSTACK=C および YSLW=0（デフォルト値）が適用されているものとします。

```
DEFINE DATA LOCAL
  1 #DATE1 (D) INIT <D'1956-12-31'>
  1 #DATE2 (D)
END-DEFINE
...
STACK TOP DATA #DATE1 /* century information is lost (year 56 is stacked)
...
INPUT #DATE2          /* 56 is assumed to be in current century -> 1956
...
/* Result: RUNTIME ERROR (UNINTENDED CENTURY CHANGE)
```

この場合、世紀情報は（意図しない値に）変更されます。 ただし、この変更は DFSTACK=C 設定によってインターセプトされます。

例 **4**：

以下の例では、現在の年は 2005 年で、パラメータの設定には DFSTACK=C および YSLW=60 が適用されているものとします。

```
DEFINE DATA LOCAL
  1 #DATE1 (D) INIT <D'2056-12-31'>
  1 #DATE2 (D)
END-DEFINE
...
STACK TOP DATA #DATE1 /* century information is lost (year 56 is stacked)
...
INPUT #DATE2          /* year sliding window determines 56 to be 1956
...
/* Result: RUNTIME ERROR (UNINTENDED CENTURY CHANGE)
```

この場合、世紀情報は年スライディングウィンドウに従って変更されます。 ただし、この変更は DFSTACK=C 設定によってインターセプトされます。

年固定ウィンドウ

このトピックの詳細については、プロファイルパラメータ YSLW の説明を参照してください。

デフォルトページタイトル用の日付フォーマット - DFTITLE パラメータ

セッション／プロファイルパラメータ DFTITLE によって、デフォルトのページタイトル（DISPLAY、WRITE、PRINT の各ステートメントの出力）の日付フォーマットが決まります。

DFTITLE=S	日付は、デリミタ付きの 2 桁の年コンポーネントで出力されます（*yy-mm-dd*）。
DFTITLE=L	日付は、デリミタ付きの 4 桁の年コンポーネントで出力されます（*yyyy-mm-dd*）。
DFTITLE=I	日付は、デリミタなしの 4 桁の年コンポーネントで出力されます（*yyyymmdd*）。

これらの各出力フォーマットに対する、年、月、日の各コンポーネントの順序、および使用するデリミタ文字は、DTFORM パラメータによって決まります。

DFTITLE パラメータは、Natural パラメータモジュール／ファイル内で、または Natural を起動するときにダイナミックに、あるいはシステムコマンド GLOBALS を使用して設定できます。 これは、ランタイム時に評価されます。

例：

この例では、DTFORM=I が適用されると仮定します。

```
WRITE 'HELLO'
END
/*
/* Date in page title if DFTITLE=S is set ...: 05-10-31
/* Date in page title if DFTITLE=L is set ...: 2005-10-31
/* Date in page title if DFTITLE=I is set ...: 20051031
```

注意: DFTITLE パラメータは、WRITE TITLE ステートメントで指定されているユーザー定義のページタイトルには効果がありません。

49 テキスト表記

INPUT、DISPLAY、WRITE、WRITE TITLE、および WRITE TRAILER の各ステートメントでは、テキスト表記を使用して、これらのステートメントとともに使用するテキストを定義できます。

ステートメントで使用するテキストの定義 - 'text' 表記

ステートメントで使用するテキスト（プロンプトメッセージなど）は、アポストロフィ（'）または引用符（"）で囲む必要があります。 アポストロフィ 2 つ（''）と引用符（"）を混同しないようにしてください。

引用符で囲んだテキストは、小文字から大文字に自動変換できます。 自動変換を無効にするには、エディタプロファイルの設定を変更します。 詳細については、（『エディタ』ドキュメントの「*全般的な情報*」の）「*エディタプロファイル*」の「*全般的なデフォルト設定*」に記載されている、［Dynamic Conversion of Lower Case］オプションの説明を参照してください。

テキスト自体には 1～72 文字を使用できますが、複数行にわたって記述することはできません。

テキスト要素は、ハイフンを使用して連結できます。

例：

```
DEFINE DATA LOCAL
1 #A(A10)
END-DEFINE

INPUT 'Input XYZ' (CD=BL) #A
WRITE '=' #A
WRITE 'Write1 ' - 'Write2 ' - 'Write3' (CD=RE)
END
```

テキスト文字列の一部としてのアポストロフィの使用

Natural プロファイルパラメータ TQ（引用符の変換）、または Natural プロファイルパラメータ CMPO のキーワードサブパラメータ TQMARK が ON に設定されていると、以下の処理が適用されます。 これはデフォルト設定です。

アポストロフィで囲まれたテキスト文字列内で 1 つのアポストロフィを表すには、2 つのアポストロフィ（''）または 1 つの引用符（"）を使用する必要があります。 どちらの表記を使用しても、単一のアポストロフィが出力されます。

引用符で囲まれたテキスト文字列内で 1 つのアポストロフィを表すには、1 つのアポストロフィを指定します。

アポストロフィの例：

```
#FIELDA = 'O''CONNOR'
#FIELDA = 'O"CONNOR'
#FIELDA = "O'CONNOR"
```

上記の 3 例すべてで、以下の結果が出力されます。

```
O'CONNOR
```

テキスト文字列の一部としての引用符の使用

Natural プロファイルパラメータ TQ（引用符の変換）、または Natural プロファイルパラメータ CMPO のキーワードサブパラメータ TQMARK が OFF に設定されていると、以下の処理が適用されます。 デフォルトの設定は ON です。

1 つのアポストロフィで囲まれたテキスト文字列内で 1 つの引用符を表すには、1 つの引用符を指定します。

引用符で囲まれたテキスト文字列内で 1 つの引用符を表すには、2 つの引用符（""）を指定します。

引用符の例：

```
#FIELDA = 'O"CONNOR'
#FIELDA = "O""CONNOR"
```

上記の 2 例すべてで、以下の結果が出力されます。

```
O"CONNOR
```

フィールド値の前に n 回出力する文字の定義 - 'c'(n) 表記

単一の文字をテキストとして複数回出力するには、以下の表記を使用します。

```
'c'(n)
```

c には出力する文字を指定し、*n* には文字を出力する回数を指定します。 *n* の最大値は、249 です。

例：

```
WRITE '*'(3)
```

文字 *c* の前後には、アポストロフィの代わりに引用符も使用できます。

50 ユーザーコメント

ユーザーコメントとは、ソースコードのステートメント間に追加または挿入する記述や説明メモのことです。これらの情報は、自分以外のプログラマによって作成または変更されたソースコードの理解や管理に特に有効です。 また、コメント開始を示す文字列は、テストのためにステートメントの機能や複数のソースコード行を一時的に無効にするために使用することもできます。

ソースコード行全体をコメントとして使用する方法

ソースコード行全体をユーザーコメントとして使用する場合、行の先頭に以下の1つを入力します。

- アスタリスクと空白（* ）
- 2 つのアスタリスク（**）
- スラッシュとアスタリスク（/*）

```
*  USER COMMENT
** USER COMMENT
/* USER COMMENT
```

例：

以下の例のように、コメント行は、ソースコードの構造を明確にするために使用することもできます。

```
** Example 'LOGICX03': BREAK option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #BIRTH (A8)
END-DEFINE
*
LIMIT 10
READ EMPLOY-VIEW BY BIRTH
  MOVE EDITED BIRTH (EM=YYYYMMDD) TO #BIRTH
  /*
  IF BREAK OF #BIRTH /6/
    NEWPAGE IF LESS THAN 5 LINES LEFT
    WRITE / '-' (50) /
  END-IF
  /*
  DISPLAY NOTITLE BIRTH (EM=YYYY-MM-DD) NAME FIRST-NAME
```

```
END-READ
END
```

ソースコード行の途中からコメントとして使用する方法

ソースコード行の途中からのみをユーザーコメントとして使用する場合、空白、スラッシュ、およびアスタリスク（/*）を入力すると、行内のこの表記以降の部分がコメントとしてマークされます。

```
ADD 5 TO #A          /* USER COMMENT
```

例：

```
** Example 'LOGICX04': IS option as format/length check
************************************************************************
DEFINE DATA LOCAL
1 #FIELDA (A10)          /* INPUT FIELD TO BE CHECKED
1 #FIELDB (N5)           /* RECEIVING FIELD OF VAL FUNCTION
1 #DATE   (A10)          /* INPUT FIELD FOR DATE
END-DEFINE
*
INPUT #DATE #FIELDA
IF #DATE IS(D)
  IF #FIELDA IS (N5)
    COMPUTE #FIELDB = VAL(#FIELDA)
    WRITE NOTITLE 'VAL FUNCTION OK' //  '=' #FIELDA '=' #FIELDB
  ELSE
    REINPUT 'FIELD DOES NOT FIT INTO N5 FORMAT'
            MARK *#FIELDA
  END-IF
ELSE
  REINPUT 'INPUT IS NOT IN DATE FORMAT (YY-MM-DD) '
          MARK *#DATE
END-IF
*
END
```

51 論理条件基準

この章では、ステートメント FIND、READ、HISTOGRAM、ACCEPT／REJECT、IF、DECIDE FOR、および REPEAT で使用できる、論理条件基準の用途と使用方法について説明します。

はじめに

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。

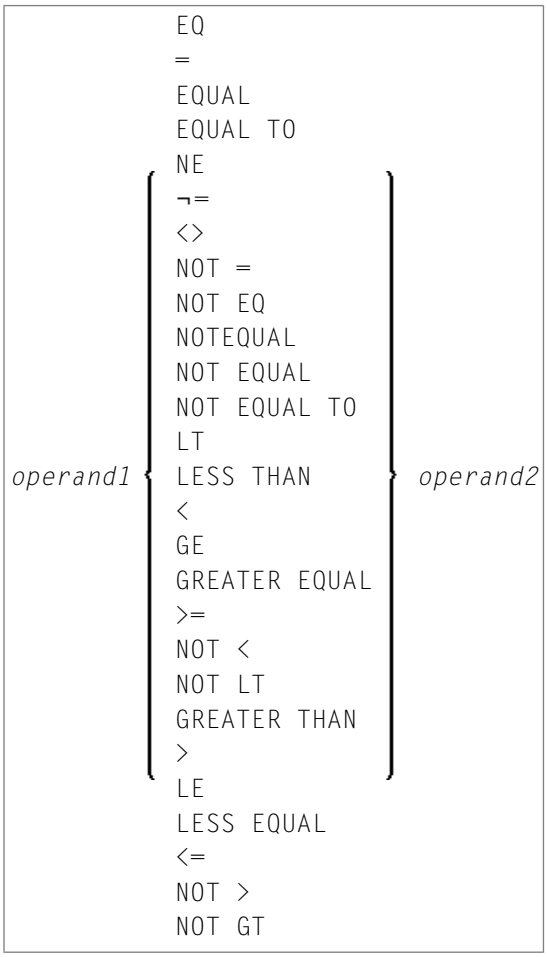
また、演算式を使用して、1 つの関係式を構成することもできます。

論理条件基準は、以下のステートメントで使用できます。

ステートメント	使用方法
FIND	論理条件基準を持つ WHERE 節を使用して、WITH 節で指定されている基本選択条件に、さらに条件を追加します。 WHERE 節で指定されている論理条件基準は、レコードの選択および読み込み後に評価されます。 WITH 節では、論理条件基準ではなく、（FIND ステートメントで記述されている）「基本検索条件」 が使用されます。
READ	論理条件基準を持つ WHERE 節を使用して、読み込んだレコードを処理するかどうかを指定します。 論理条件基準は、レコードの読み込み後に評価されます。
HISTOGRAM	論理条件基準を持つ WHERE 節を使用して、読み込んだ値を処理するかどうかを指定します。 論理条件基準は、値の読み込み後に評価されます。
ACCEPT/REJECT	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。
IF	論理条件の基準を使用して、ステートメントの実行を制御します。
DECIDE FOR	論理条件の基準を使用して、ステートメントの実行を制御します。
REPEAT	REPEAT ステートメントの UNTIL 節または WHILE 節に、いつ処理ループを終了するのかを決定する論理条件基準を指定します。

関係式

構文：



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	C	S	A		N	E	A	U	N	P	I	F	B	D	T	L		G	O	可	可
operand2	C	S	A		N	E	A	U	N	P	I	F	B	D	T	L		G	O	可	不可

上記オペランド定義テーブルの詳細については、『ステートメント』ドキュメントの「*構文記号およびオペランド定義テーブル*」を参照してください。

上記の表の 「構文要素」 の 「E」 は、算術式を表します。つまり、関係式のオペランドとして算術式を指定できることを意味します。 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。

関係式の例：

```
IF NAME = 'SMITH'
IF LEAVE-DUE GT 40
IF NAME = #NAME
```

関係式における配列の比較については、「*配列の処理*」を参照してください。

注意: 浮動小数点のオペランドを使用すると、浮動小数点で比較が実行されます。浮動小数点数自体の精度に制限があるため、数値と浮動小数点形式との変換を行うときに、切り上げ／切り捨てエラーを回避することはできません。

論理条件における算術式

以下の例は、論理条件で算術式をどのように使用するかを示しています。

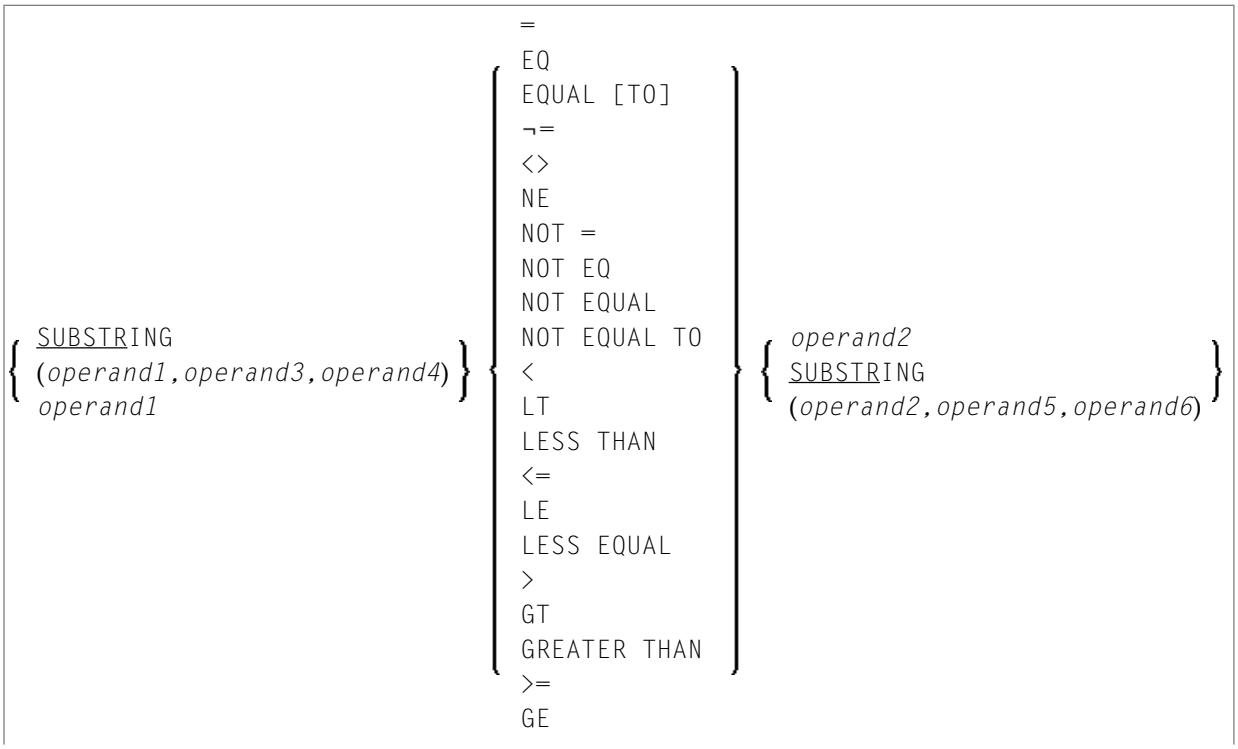
```
IF #A + 3 GT #B - 5 AND #C * 3 LE #A + #B
```

論理条件におけるハンドル

関係式のオペランドがハンドルの場合、演算子 EQUAL および NOT EQUAL のみを使用できます。

関係式における **SUBSTRING** オプション

構文：



GREATER EQUAL

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	C	S	A		N		A	U				B							可	可
operand2	C	S	A		N		A	U				B							可	不可
operand3	C	S							N	P	I	B							可	不可
operand4	C	S							N	P	I								可	不可
operand5	C	S							N	P	I								可	不可
operand6	C	S							N	P	I								可	不可

SUBSTRING オプションを使用すると、英数字、バイナリ、または Unicode の各フィールドの一*部*を比較できます。 フィールド名（*operand1*）の後にまず開始位置（*operand3*）を指定し、次に、フィールドの比較する部分の長さ（*operand4*）を指定します。

また、フィールド値を別のフィールド値の一部と比較することもできます。 フィールド名（*operand2*）の後にまず開始位置（*operand5*）を指定し、次に、フィールド *operand1* の比較する部分の長さ（*operand6*）を指定します。

両方の形式を組み合わせることもできます。つまり、SUBSTRING を *operand1* と *operand2* の両方に指定できます。

例：

以下の式は、フィールド #A の値の 5～12 桁目とフィールド #B の値を比較しています。

```
SUBSTRING(#A,5,8) = #B
```

- 5 が開始位置で、8 が長さです。

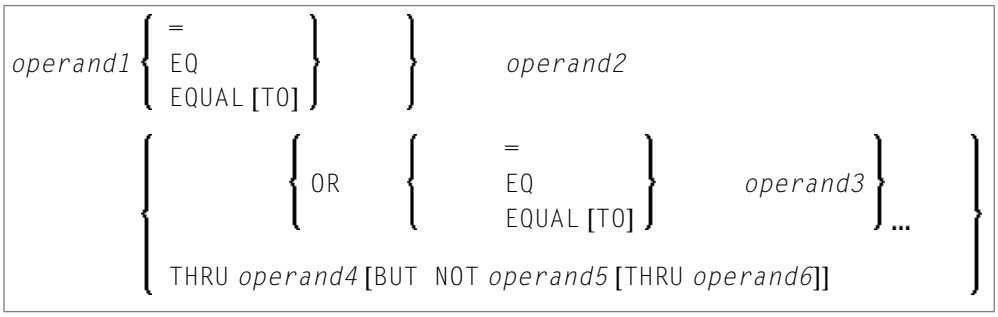
以下の式は、フィールド #A の値とフィールド #B の値の 3～6 桁目を比較しています。

```
#A = SUBSTRING(#B,3,4)
```

注意: *operand3*／*operand5* を省略すると、開始位置は 1 とみなされます。 *operand4*／*operand6* 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みなされます。

拡張関係式

構文：



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	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	可	不可
operand2	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	可	不可
operand3	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	可	不可
operand4	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	可	不可
operand5	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	可	不可
operand6	C	S	A		N*	E	A	U	N	P	I	F	B	D	T			G	O	可	不可

* 算術関数およびシステム変数は使用できますが、 ブレイク関数は使用できません。

operand3 は、以下のように、MASK オプションまたは SCAN オプションを使用して指定することもできます。

```
MASK (mask-definition) [operand]
MASK operand
SCAN operand
```

これらのオプションの詳細については、「*MASK オプション*」および「*SCAN オプション*」を参照してください。

例：

```
IF #A = 2 OR = 4 OR = 7
IF #A = 5 THRU 11 BUT NOT 7 THRU 8
```

MASK オプション

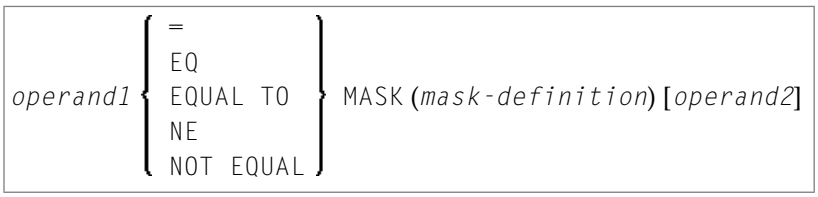
MASK オプションを使用すると、フィールドの特定の位置を特定の内容でチェックできます。

以下では次のトピックについて説明します。

- 定数マスク
- 変数マスク
- マスク内の文字
- マスク長
- 日付チェック
- 定数または変数の内容に対するチェック
- 範囲チェック
- パック型数値データまたはアンパック型数値データのチェック

定数マスク

構文：



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	C	S	A		N		A	U	N	P									可	不可
operand2	C	S					A	U	N	P			B						可	不可

Operand2 は、*mask-definition* に少なくとも 1 つの X が指定されている場合にのみ使用できます。 *Operand1* および *operand2* は、互換性のあるフォーマットである必要があります。

- *operand1* がフォーマット A の場合、*operand2* はフォーマット A、B、N、または U である必要があります。

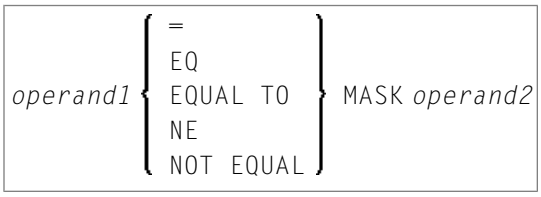
- *operand1* がフォーマット U の場合、*operand2* はフォーマット A、B、N、または U である必要があります。
- *operand1* がフォーマット N または P の場合、*operand2* はフォーマット N または P である必要があります。

mask-definition 内の X は、*operand1* および *operand2* の対応する位置を選択して比較します。

変数マスク

定数 *mask-definition*（上記参照）の他に、変数マスク定義を使用することもできます。

構文：



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	C	S	A		N		A	U	N	P								可	不可
operand2		S					A	U										可	不可

operand2 の内容はマスク定義とみなされます。 *operand2* 内の末尾の空白は無視されます。

- *operand1* がフォーマット A、N、または P の場合、*operand2* はフォーマット A である必要があります。
- *operand1* がフォーマット U の場合、*operand2* はフォーマット U である必要があります。

マスク内の文字

マスク定義（定数マスクのマスク定義、および変数マスクの *operand2*）には、以下の文字を使用できます。

文字	意味
. または ? または _	ピリオド、疑問符、または下線は、該当する 1 桁をチェックしないことを示します。
* または %	アスタリスクまたはパーセント記号は、任意の桁数をチェックしないことを示すために使用します。
/	スラッシュ（/）は、値が特定の文字（または文字列）で終わっているかどうかをチェックするために使用します。 例えば、フィールドの最後が E であるか、または最後の E の後が空白以外に何もない場合、以下の条件は真になります。 `IF #FIELD = MASK (*'E'/)`
A	該当する桁が英字（大文字または小文字）かどうかをチェックします。
'*c*'	1 つ以上の桁が、アポストロフィで囲まれている文字列かどうかをチェックします。アポストロフィを 2 つ使用すると、単一のアポストロフィをチェックできます。 16 進数値で H'40'（空白）未満の英数字は使用できません。
C	該当する桁が英字（大文字または小文字）、数字、または空白かどうかをチェックします。
DD	該当する 2 桁が有効な日の表記（01～31。MM および YY/YYYY が指定されていれば、その値に依存。「*日付チェック*」も参照）かどうかをチェックします。
H	該当する桁が 16 進数表記（A～F、0～9）かどうかをチェックします。
JJJ	該当する桁が有効なユリウス日、つまり年における日数（001～366。YY/YYYY が指定されていれば、その値に依存。「*日付チェック*」も参照）かどうかをチェックします。
L	該当する桁が小文字の英字（a～z）かどうかをチェックします。
MM	該当する桁が有効な月（01～12）かどうかをチェックします。「*日付チェック*」も参照してください。
N	該当する桁が数値かどうかをチェックします。
n...	1 つ（以上）の桁が 0～*n* の範囲内の数値かどうかをチェックします。
*n*1-*n*2 または *n*1:*n*2	該当する桁が *n*1-1～*n*2 の範囲内の数値かどうかをチェックします。2. *n*1 および *n*2 は同じ長さである必要があります。
P	該当する桁が表示可能な文字（U、L、N、または S）かどうかをチェックします。
S	該当する桁が特殊文字かどうかをチェックします。
U	該当する桁が大文字の英字（A～Z）かどうかをチェックします。
X	該当する桁が、マスク定義に続く値（*operand2*）の同じ位置の値と同じかどうかをチェックします。 X は、変数マスク定義では意味も持たないため、使用できません。
YY	該当する 2 桁が有効な年（00～99）かどうかをチェックします。「*日付チェック*」も参照してください。
YYYY	該当する 4 桁が有効な年（0000～2699）かどうかをチェックします。 有効な年の範囲を 1582～2699 に制限するには、COMPOPT オプション MASKCME=ON を使用します。「*日付*

文字	意味
	チェック」も参照してください。 プロファイルパラメータ MAXYEAR が 9999 に設定されている場合、年の上限は 9999 です。
Z	該当する桁の左側のハーフバイトが 16 進数の A～Fで、右側のハーフバイトが 16 進数の 0～9 かどうかをチェックします。 これは、負数の数値に対しても正しくチェックできます。 数値の符号は最後の桁に格納されるため、その桁は 16 進数の非数値とみなされます。したがって、（該当する桁が数値かどうかをチェックする）N では、負数の数値を正しくチェックできません。 マスク内では Z は、チェック対象の一連の数字ごとに一度だけ使用します。

マスク長

マスクの長さにより、チェックが必要な位置の数が決まります。

例：

```
DEFINE DATA LOCAL
1 #CODE (A15)
END-DEFINE
...
IF #CODE = MASK (NN'ABC'....NN)
...
```

上記の例では、#CODE の最初の 2 桁に対し、数値かどうかをチェックしています。 その後の 3 桁は、定数 ABC かどうかをチェックしています。 その次の 4 桁はチェックしていません。 10 桁目および 11 桁目は、数値かどうかをチェックしています。 12～15 桁目はチェックしていません。

日付チェック

指定するマスク内でチェックできる日付は 1 つのみです。

日付に対するチェックでマスクに日（DD）を指定して月（MM）を指定しない場合、現在の月を想定してチェックが行われます。

日付に対するチェックでマスクに日（DD）またはユリウス日（JJJ）を指定して年（YY または YYYY）を指定しない場合、現在の年を想定してチェックが行われます。

日付を 2 桁の年（YY）でチェックするときにスライディングウィンドウも固定ウィンドウも設定されていない場合、現在の世紀を想定してチェックが行われます。 スライディングウィンドウまたは固定ウィンドウの詳細については、『*パラメータリファレンス*』に記載されている、プロファイルパラメータ YSLW の説明を参照してください。

例 **1**：

```
MOVE 1131 TO #DATE (N4)
IF #DATE = MASK (MMDD)
```

この例では、月と日の有効性をチェックしています。 月に対する値（11）は有効とみなされますが、11 月には 30 日しかないため、日に対する値（31）は無効とみなされます。

例 **2**：

```
IF #DATE(A8) = MASK (MM'/'DD'/'YY)
```

この例では、フィールド #DATE の内容がフォーマット MM/DD/YY（月／日／年）の日付として有効かどうかをチェックしています。

例 **3**：

```
IF #DATE (A4) = MASK (19-20YY)
```

この例では、フィールド #DATE の内容が 19～20 の範囲内の 2 桁の数字とそれに続く有効な 2 桁の年（00～99）かどうかをチェックしています。 Natural では、上記の方法で世紀を割り当てています。

注意: 明白なことですが、上記のマスクでは、年の整合性とは関係なく数値範囲 19～20 をチェックしているため、1900～2099 の範囲内で有効な年をチェックすることはできません。 年の範囲をチェックするには、日付の整合性をチェックするコードと範囲の有効性をチェックするコードを別個に記述します。

```
IF #DATE (A10) = MASK (YYYY'-'MM'-'DD) AND #DATE = MASK (19-20)
```

定数または変数の内容に対するチェック

マスクチェックに対する値として定数または変数を使用する場合、この値（*operand2*）は *mask-definition* の直後に指定する必要があります。

Operand2 は、少なくともマスクと同じ長さである必要があります。

マスクでは、チェックする桁には X、チェックしない桁にはピリオド（.）、疑問符（?）、または下線（_）を指定します。

例：

```
DEFINE DATA LOCAL
1 #NAME (A15)
END-DEFINE
...
IF #NAME = MASK (..XX) 'ABCD'
...
```

上記の例では、フィールド #NAME の 3～4 桁目が CD かどうかをチェックしていいます。 1 桁目および 2 桁目はチェックされません。

マスクの長さにより、チェックが必要な位置の数が決まります。 マスク操作で使用する任意のフィールドまたは定数に対し、マスクは左詰めで適用されます。 式の右側のフィールド（または定数）のフォーマットは、式の左側のフィールドのフォーマットと同じである必要があります。

チェックするフィールド（*operand1*）がフォーマット A の場合、使用する定数（*operand2*）はアポストロフィで囲む必要があります。 フィールドが数値の場合、使用する値は数値定数、数値データベースフィールドの値、またはユーザー定義変数である必要があります。

どちらの場合でも、マスク内の X インジケータが示す位置に該当しない文字／数値は無視されます。

指定された位置の値が両方とも同じ場合、MASK 操作の結果は真になります。

例：

```
** Example 'LOGICX01': MASK option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
END-DEFINE
*
HISTOGRAM EMPLOY-VIEW CITY
  IF CITY =
MASK (....XX) '....NN'


    DISPLAY NOTITLE CITY *NUMBER
  END-IF
END-HISTOGRAM
*
END
```

上記の例では、フィールド CITY の 5～6 桁目が両方とも文字 N の場合、当該レコードは受け入れられます。

範囲チェック

範囲チェックを実行する場合、指定した変数を検証する桁数は、マスクに指定する値の精度で定義します。 例えば、(...193...)というマスクを指定すると、4〜6桁目が000〜193の範囲内の3桁の数値かどうかが検証されます。

マスク定義の他の例

- 以下の例では、#NAMEの各文字が英字かどうかがチェックされます。

```
IF #NAME (A10) = MASK (AAAAAAAAAA)
```

- 以下の例では、#NUMBERの4〜6桁目が数値かどうかがチェックされます。

```
IF #NUMBER (A6) = MASK (...NNN)
```

- 以下の例では、#VALUEの4〜6桁目が値123かどうかがチェックされます。

```
IF #VALUE(A10) = MASK (...'123')
```

- 以下の例では、#LICENSEがNY-で始まるライセンス番号で、最後の5文字が#VALUEの最後の5文字と同じかどうかがチェックされます。

```

DEFINE DATA LOCAL
1 #VALUE(A8)
1 #LICENSE(A8)
END-DEFINE
INPUT 'ENTER KNOWN POSITIONS OF LICENSE PLATE:' #VALUE
IF #LICENSE = MASK ('NY-'XXXXX) #VALUE
```

- 以下の条件は、NATで始まりALで終わる値は、NATとALの間に他の文字が何文字あっても、すべて真になります。これには、NATALだけでなく、NATURALやNATIONALITYなどの値も含まれます。

```
MASK('NAT'*'AL')
```

パック型数値データまたはアンパック型数値データのチェック

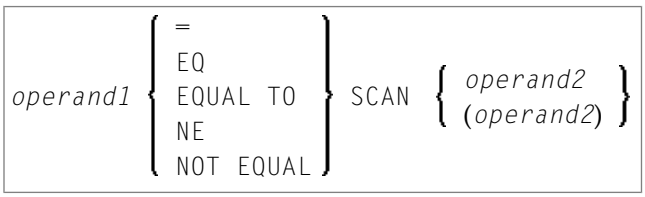
レガシアプリケーションでは、英数字フィールドやバイナリフィールドを再定義したパック型またはアンパック型の数値変数を頻繁に使用しています。 パック型変数またはアンパック型変数を割り当てや計算で使用すると、エラーや予測できない結果が生じる可能性があるため、このような再定義はしないことをお勧めします。再定義した変数を使用する前に内容を検証するには、桁数 - 1 個の N オプション（「マスク内の文字」を参照）を使用し、その後に一度だけ単一の Z オプションを指定します。

例：

```
IF #P1 (P1) = MASK (Z)
IF #N4 (N4) = MASK (NNNZ)
IF #P5 (P5) = MASK (NNNNZ)
```

SCAN オプション

構文：



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	C	S	A		N		A	U	N	P									可	不可
operand2	C	S					A	U					B*						可	不可

operand1 がフォーマット A または U の場合にのみ、***operand2* にバイナリを使用できます。*operand1* がフォーマット U で、*operand2* がフォーマット B の場合、*operand2* の長さは偶数である必要があります。

SCAN オプションは、フィールド内の特定の値をスキャンするために使用します。

SCAN オプション（*operand2*）に使用する文字は、英数字定数または Unicode 定数（アポストロフィで区切られた文字列）、英数字データベースフィールドまたは Unicode データベースフィールドの値、あるいはユーザー定義変数として指定する必要があります。

注意: *operand1* および *operand2* の末尾の空白は、自動的に除去されます したがって、SCAN オプションは、末尾に空白のある値のスキャンには使用できません。 *operand1* およ

び *operand2* の先頭または途中の空白は保持されます。 *operand2* がすべて空白の場合、*operand1* の値に関係なく、スキャンは成功とみなされます。スキャン操作で末尾の空白をスキャン対象とする場合は、EXAMINE FULL ステートメントの使用を検討してください。

スキャンするフィールド（*operand1*）には、フォーマット A、N、P、または U を使用できます。 SCAN 操作には、EQ（等しい）または NE（等しくない）の各演算子を指定できます。

SCAN 操作に使用する文字列の長さは、スキャンされるフィールドの長さ未満である必要があります。 指定した文字列の長さが、スキャンするフィールドの長さと同じ場合、SCAN ではなく EQUAL 演算子を使用する必要があります。

SCAN オプションの例：

```
** Example 'LOGICX02': SCAN option in logical condition
************************************************************************
DEFINE DATA
LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
*
1 #VALUE   (A4)
1 #COMMENT (A10)
END-DEFINE
*
INPUT 'ENTER SCAN VALUE:' #VALUE
LIMIT 15
*
HISTOGRAM EMPLOY-VIEW FOR NAME
  RESET #COMMENT
  IF NAME = SCAN #VALUE
    MOVE 'MATCH' TO #COMMENT
  END-IF
  DISPLAY NOTITLE NAME *NUMBER #COMMENT
END-HISTOGRAM
*
END
```

プログラム LOGICX02 の出力：

```
ENTER SCAN VALUE:
```

15 件の名前を LL でスキャンした結果、3 件一致した例：

```
        NAME           NMBR      #COMMENT
-------------------- --------- ----------

ABELLAN                      1 MATCH
ACHIESON                     1
ADAM                         1
ADKINSON                     8
AECKERLE                     1
AFANASSIEV                   2
AHL                          1
AKROYD                       1
ALEMAN                       1
ALESTIA                      1
ALEXANDER                    5
ALLEGRE                      1 MATCH
ALLSOP                       1 MATCH
ALTINOK                      1
ALVAREZ                      1
```

論理条件基準における BREAK

BREAKオプションでは、フィールドの現在の値または値の一部と、処理ループで前回通過した、同じフィールドの値とを比較できます。

構文：

```
BREAK [OF] operand1 [/n/]
```

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	U	N	P	I	F	B	D	T	L				可	不可

構文要素の説明：

operand1	チェックするコントロールフィールドを指定します。 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。
/n/	-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、フォーマット A、B、N、または P のオペランドに対してのみ使用できます。 指定した位置のフィールドの値が変わると、BREAK 操作の結果は真になります。 AT END OF DATA 条件が発生すると、BREAK 操作の結果は偽になります。

	例： 以下の例では、フィールド FIRST-NAME の最初の文字が変わっているかどうかがチェックされます。 `BREAK FIRST-NAME /1/` このオプションには、Natural システム関数を使用できません（AT BREAK ステートメントでは使用可能）。

BREAK オプションの例：

```
** Example 'LOGICX03': BREAK option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #BIRTH (A8)
END-DEFINE
*
LIMIT 10
READ EMPLOY-VIEW BY BIRTH
  MOVE EDITED BIRTH (EM=YYYYMMDD) TO #BIRTH
  /*
  IF BREAK OF #BIRTH /6/
    NEWPAGE IF LESS THAN 5 LINES LEFT
    WRITE / '-' (50) /
  END-IF
  /*
  DISPLAY NOTITLE BIRTH (EM=YYYY-MM-DD) NAME FIRST-NAME
END-READ
END
```

プログラム LOGICX03 の出力：

```
  DATE            NAME              FIRST-NAME
   OF
 BIRTH
---------- -------------------- --------------------

1940-01-01 GARRET               WILLIAM
1940-01-09 TAILOR               ROBERT
1940-01-09 PIETSCH              VENUS
1940-01-31 LYTTLETON            BETTY
```

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

1940-02-02 WINTRICH             MARIA
1940-02-13 KUNEY                MARY
1940-02-14 KOLENCE              MARSHA
1940-02-24 DILWORTH             TOM

------------------------------------------------

1940-03-03 DEKKER               SYLVIA
1940-03-06 STEFFERUD            BILL
```

IS オプション - 値のフォーマットおよび長さのチェック

構文：

operand1 IS (*format*)

このオプションは、英数字フィールドまたは Unicode フィールド（*operand1*）の値を特定の別のフォーマットに変換できるかどうかをチェック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											参照	ダイナミック定義
operand1	C	S	A		N		A	U										可	不可

チェックできる *format* は、以下のとおりです。

N*n.m*	長さが *n.m* の数値
F*n*	長さが *n* の浮動小数点
D	日付。 *dd-mm-yy*、*dd-mm-yyyy*、*ddmmyyyy*（*dd* = 日、*mm* = 月、*yy* または *yyyy* = 年）の各日付フォーマットを使用できます。 年、月、日の各コンポーネントの順序、およびコンポーネント間のデリミタ文字は、プロファイルパラメータ DTFORM（『パラメータリファレンス』を参照）によって決まります。
T	時刻（デフォルトの時刻表示フォーマットに依存）
P*n.m*	長さが *n.m* のパック型数値
I*n*	長さが *n* の整数値

チェック時は、*operand1* の先頭および末尾の空白は無視されます。

IS オプションは、例えば、算術関数 VAL（英数字フィールドから数値を抽出）を実行する前にフィールドの内容をチェックして、ランタイムエラーが発生しないようにするために使用できます。

注意: IS オプションで可能なのは、英数字フィールドの値が特定の「フォーマット」かどうかのチェックではなく、その「フォーマット」に*変換*できるかどうかのチェックです。 値が特定のフォーマットかどうかをチェックするには、MASK オプションを使用します。

IS オプションの例：

```
** Example 'LOGICX04': IS option as format/length check
************************************************************************
DEFINE DATA LOCAL
1 #FIELDA (A10)          /* INPUT FIELD TO BE CHECKED
1 #FIELDB (N5)           /* RECEIVING FIELD OF VAL FUNCTION
1 #DATE   (A10)          /* INPUT FIELD FOR DATE
END-DEFINE
*
INPUT #DATE #FIELDA
IF #DATE IS(D)
  IF #FIELDA IS (N5)
    COMPUTE #FIELDB = VAL(#FIELDA)
    WRITE NOTITLE 'VAL FUNCTION OK' //  '=' #FIELDA '=' #FIELDB
  ELSE
    REINPUT 'FIELD DOES NOT FIT INTO N5 FORMAT'
            MARK *#FIELDA
  END-IF
ELSE
  REINPUT 'INPUT IS NOT IN DATE FORMAT (YY-MM-DD) '
          MARK *#DATE
END-IF
*
END
```

プログラム LOGICX04 の出力：

```
#DATE 150487    #FIELDA









INPUT IS NOT IN DATE FORMAT (YY-MM-DD)
```

論理変数の評価

構文：

operand1

このオプションは、論理変数（フォーマットL）はとともに使用します。論理変数の値は、TRUE または FALSE になります。 *operand1* には、使用する論理変数名を指定します。

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	参照	ダイナミック定義
operand1	C	S	A	L	不可	不可

論理変数の例：

```
** Example 'LOGICX05': Logical variable in logical condition
************************************************************************
DEFINE DATA LOCAL
1 #SWITCH (L)  INIT <true>
1 #INDEX  (I1)
END-DEFINE
*
FOR #INDEX 1 5
  WRITE NOTITLE #SWITCH (EM=FALSE/TRUE) 5X 'INDEX =' #INDEX
  WRITE NOTITLE #SWITCH (EM=OFF/ON) 7X 'INDEX =' #INDEX
  IF #SWITCH
    MOVE FALSE TO #SWITCH
  ELSE
    MOVE TRUE TO #SWITCH
  END-IF
  /*
  SKIP 1
END-FOR
END
```

プログラム LOGICX05 の出力：

```
TRUE     INDEX =    1
ON       INDEX =    1

FALSE    INDEX =    2
OFF      INDEX =    2

TRUE     INDEX =    3
ON       INDEX =    3

FALSE    INDEX =    4
OFF      INDEX =    4

TRUE     INDEX =    5
ON       INDEX =    5
```

MODIFIED オプション

構文：

operand1 [NOT] MODIFIED

このオプションは、属性がダイナミックに割り当てられているフィールドの内容が、INPUT ステートメントの実行中に変更されているかどうかを判断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	参照	ダイナミック定義
operand1	S A	C	不可	不可

INPUT ステートメント内で参照される属性制御変数には、端末にマップが転送される時点では常にステータス NOT MODIFIED が割り当てられています。

属性制御変数を参照しているフィールドの内容が変更されると、属性制御変数にステータス MODIFIED が割り当てられます。 複数のフィールドが同じ属性制御変数を参照している場合、これらのフィールドのどれかが変更されると、この変数は MODIFIED に設定されます。

operand1 が配列の場合、配列要素の少なくとも 1 つにステータス MODIFIED が割り当てられていると、結果は真になります（OR 演算）。

Natural プロファイルパラメータ CVMIN を使用すると、対応するフィールドの値が*同一*の値で上書きされた場合にも属性制御変数にステータス MODIFIED を割り当てるかどうかを指定できます。

MODIFIED オプションの例：

```
** Example 'LOGICX06': MODIFIED option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 #ATTR (C)
1 #A    (A1)
1 #B    (A1)
END-DEFINE
*
MOVE (AD=I) TO #ATTR
*
INPUT (CV=#ATTR) #A #B
IF #ATTR NOT MODIFIED
  WRITE NOTITLE 'FIELD #A OR #B HAS NOT BEEN MODIFIED'
END-IF
*
IF #ATTR MODIFIED
  WRITE NOTITLE 'FIELD #A OR #B HAS BEEN MODIFIED'
END-IF
*
END
```

プログラム LOGICX06 の出力：

```
#A   #B
```

任意の値を入力して Enter キーを押すと、以下のような出力が表示されます。

```
FIELD #A OR #B HAS BEEN MODIFIED
```

SPECIFIED オプション

構文：

operand1 [NOT] SPECIFIED

このオプションは、呼び出されたオブジェクト（サブプログラムまたは外部サブルーチン）のオプションパラメータが、呼び出し元オブジェクトから値を受け取っているかどうかをチェックするために使用します。

オプションパラメータとは、呼び出されるオブジェクトの DEFINE DATA PARAMETER ステートメント内でキーワード OPTIONAL を使用して定義されているフィールドのことです。フィールドが

OPTIONAL として定義されている場合、呼び出し元オブジェクトからこのフィールドに値を渡すことはできますが、必須ではありません。

呼び出し元ステートメントでは、表記 *nX* を使用して、パラメータに値を渡さないことを示します。

値を受け取っていないオプションパラメータを処理しようとすると、ランタイムエラーが発生します。 このようなエラーを回避するには、呼び出されたオブジェクト内で SPECIFIED オプションを使用してオプションパラメータに値が渡されているかどうかを確認し、値が渡されている場合にのみ処理するようにします。

Parameter-name は、呼び出されたオブジェクトの DEFINE DATA PARAMETER ステートメントで指定されているパラメータ名です。

フィールドが OPTIONAL として定義されていない場合、SPECIFIED 条件は常に TRUE になります。

SPECIFIED オプションの例：

呼び出し元プログラム：

```
** Example 'LOGICX07': SPECIFIED option in logical condition
************************************************************************
DEFINE DATA LOCAL
1 #PARM1 (A3)
1 #PARM3 (N2)
END-DEFINE
*
#PARM1 := 'ABC'
#PARM3 := 20
*
CALLNAT 'LOGICX08' #PARM1 1X #PARM3
*
END
```

呼び出されたサブプログラム：

```
** Example 'LOGICX08': SPECIFIED option in logical condition
************************************************************************
DEFINE DATA PARAMETER
1 #PARM1 (A3)
1 #PARM2 (N2) OPTIONAL
1 #PARM3 (N2) OPTIONAL
END-DEFINE
*
WRITE '=' #PARM1
*
IF #PARM2 SPECIFIED
```

```
  WRITE '#PARM2 is specified'
  WRITE '=' #PARM2
ELSE
  WRITE '#PARM2 is not specified'
* WRITE '=' #PARM2                /* would cause runtime error NAT1322
END-IF
*
IF #PARM3 NOT SPECIFIED
  WRITE '#PARM3 is not specified'
ELSE
  WRITE '#PARM3 is specified'
  WRITE '=' #PARM3
END-IF
END
```

プログラム LOGICX07 の出力：

```
Page      1                                                  04-12-15  11:25:41

#PARM1: ABC
#PARM2 is not specified
#PARM3 is specified
#PARM3:  20
```

論理条件基準内で使用するフィールド

データベースフィールドおよびユーザー定義変数は、論理条件基準を構成するために使用できます。 マルチプルバリューフィールドであるデータベースフィールド、またはピリオディックグループに含まれているデータベースフィールドも使用できます。 マルチプルバリューフィールドに値の範囲が指定されている、またはピリオディックグループにオカレンスの範囲が指定されている場合、検索値がその範囲の値／オカレンス内で検出されると、条件は真になります。

使用する値はそれぞれ、式の反対側で使用されているフィールドと互換性がある必要があります。10進数表記は、数値フィールドに対する値にのみ指定します。また、値の小数部の桁数は、フィールドに定義されている小数部の桁数と一致させる必要があります。

オペランドのフォーマットが異なる場合、2番目のオペランドが最初のオペランドのフォーマットに変換されます。

以下の表に、論理条件で一緒に使用できるオペランドのフォーマットを示します。

operand1	*operand2*												
	A	**U**	**B***n* (*n*=<4)	**B***n* (*n*>=5)	**D**	**T**	**I**	**F**	**L**	**N**	**P**	**GH**	**OH**
A	○	○	○	○									
U	○	○	[2]	[2]									
B*n* (*n*=<4)	○	○	○	○	○	○	○	○		○	○		
B*n* (*n*>=5)	○	○	○	○									
D			○		○	○	○	○		○	○		
T			○		○	○	○	○		○	○		
I			○		○	○	○	○		○	○		
F			○		○	○	○	○		○	○		
L													
N			○		○	○	○	○		○	○		
P			○		○	○	○	○		○	○		
GH [1]												○	
OH [1]													○

注意:

1. 1) ここで、GH は GUI ハンドルを表し、OH はオブジェクトハンドルを表します。
2. 2) バイナリ値には Unicode コードポイントが格納されているものとします。また、比較は、2 つの Unicode 値の比較として実行されます。 バイナリフィールドの長さは偶数である必要があります。

2つの値を英数字値として比較する場合、長い方の値と同じ長さにするために、末尾に空白を追加して短い方の値を拡張するものとします。

2つの値をバイナリ値として比較する場合、長い方の値と同じ長さにするために、先頭にバイナリゼロを挿入して短い方の値を拡張するものとします。

2 つの値を Unicode 値として比較する場合、両方の値の末尾の空白を削除してから、ICU 照合アルゴリズムを使用して 2 つの値を比較します。『*Unicode とコードページのサポート*』ドキュメントの「*論理条件の基準*」も参照してください。

比較の例：

```
A1(A1) := 'A'
A5(A5) := 'A    '
B1(B1) := H'FF'
B5(B5) := H'00000000FF'
U1(U1) := UH'00E4'
U2(U2) := UH'00610308'
IF A1 = A5 THEN ...                  /* TRUE
IF B1 = B5 THEN ...                  /* TRUE
IF U1 = U2 THEN ...                  /* TRUE
```

配列とスカラ値を比較する場合、配列の各要素がスカラ値と比較されます。 少なくとも 1 つの配列要素が条件を満たしていると、結果は真になります（OR 演算）。

配列と配列を比較する場合、配列の各要素が、もう一方の配列の対応する要素と比較されます。すべての要素の比較が条件を満たしている場合にのみ、結果は真になります（AND 演算）。

「*配列の処理*」も参照してください。

注意: Adabas フォネティックディスクリプタは、論理条件には使用できません。

論理条件基準の例：

```
FIND EMPLOYEES-VIEW WITH CITY = 'BOSTON' WHERE SEX = 'M'
READ EMPLOYEES-VIEW BY NAME WHERE SEX = 'M'
ACCEPT IF LEAVE-DUE GT 45
IF #A GT #B THEN COMPUTE #C = #A + #B
REPEAT UNTIL #X = 500
```

複雑な論理式における論理演算子

論理条件基準は、ブール演算子 AND、OR、および NOT を使用して組み合わせることができます。カッコを使用して論理グループを示すこともできます。

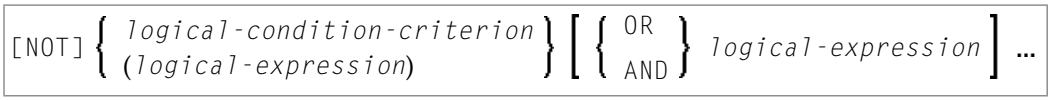
演算子は、以下の順序で評価されます。

優先順位	演算子	意味
1	( )	カッコ
2	NOT	否定
3	AND	AND 操作
4	OR	OR 操作

以下の*論理条件基準*を論理演算子で組み合わせることにより、複雑な*論理式*を作成できます。

- 関係式
- 拡張関係式
- MASK オプション
- SCAN オプション
- BREAK オプション

*論理式*の構文は以下のとおりです。



論理式の例：

```
FIND STAFF-VIEW WITH CITY = 'TOKYO'
     WHERE BIRTH GT 19610101 AND SEX = 'F'
IF NOT (#CITY = 'A' THRU 'E')
```

論理式における配列の比較については、「*配列の処理*」を参照してください。

注意: 複数の論理条件基準がANDで組み合わされている場合、それらの条件のいずれかが偽になると即座に評価は終了します。

52 演算割り当てのルール

フィールドの初期化

算術演算のオペランドとして使用するフィールド（ユーザー定義変数またはデータベースフィールド）は、以下のフォーマットのいずれかで定義する必要があります。

フォーマット	
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
D	日付
T	時刻

注意: レポーティングモードの場合、算術演算のオペランドとして使用するフィールドをあらかじめ定義しておく必要があります。算術演算の結果フィールドとして使用するユーザー定義変数またはデータベースフィールドについては、あらかじめ定義しておく必要はありません。

DEFINE DATA ステートメントで定義されているユーザー定義変数およびデータベースフィールドはすべて、実行するためにプログラムが呼び出されるときに、ゼロまたは空白で適切に初期化されます。

データ転送

データ転送は、MOVE ステートメントまたは COMPUTE ステートメントを使用して実行します。 次の表は、オペランドで使用可能な、データ転送で互換性のあるフォーマットをまとめたものです。

送出側のフィールドフォーマット	受け取り側のフィールドフォーマット												
	N または P	A	U	B*n* (*n*<5)	B*n* (*n*>4)	I	L	C	D	T	F	G	O
N または P	○	[2]	[14]	[3]	-	○	-	-	-	○	○	-	-
A	-	○	[13]	[1]	[1]	-	-	-	-	-	-	-	-
U	-	[11]	○	[12]	[12]	-	-	-	-	-	-	-	-
B*n* (*n*<5)	[4]	[2]	[14]	[5]	[5]	○	-	-	-	○	○	-	-
B*n* (*n*>4)	-	[6]	[15]	[5]	[5]	-	-	-	-	-	-	-	-

I	○	[2]	[14]	[3]	-	○	-	-	-	○	○	-	-	
L	-	[9]	[16]	-	-	-	○	-	-	-	-	-	-	
C	-	-	-	-	-	-	-	○	-	-	-	-	-	
D	○	[9]	[16]	○	-	○	-	-	○	[7]	○	-	-	
T	○	[9]	[16]	○	-	○	-	-	[8]	○	○	-	-	
F	○	[9] [10]	[10] [16]	[3]	-	○	-	-	-	○	○	-	-	
G	-	-	-	-	-	-	-	-	-	-	-	○	-	
O	-	-	-	-	-	-	-	-	-	-	-	-	○	

上記の意味は次に示すとおりです。

○	データ転送の互換性があることを示します。
-	データ転送の互換性がないことを示します。
[ ]	角カッコ [ ] は、下記の対応するデータ転送ルールを参照します。

データ変換

データ値の変換には、以下のルールが適用されます。

1. 英数字からバイナリ
   値は左から右に 1 バイトずつ移動します。 定義されている長さおよび指定されているバイト数に応じて、変換後の値は切り捨てられるか、または末尾に空白文字が追加されます。
2. （**N**、**P**、**I**）およびバイナリ（長さ **1**～**4**）から英数字
   値はアンパック形式に変換され、左詰めで英数字フィールドに転送されます。つまり、先行ゼロは除去され、フィールドには末尾の空白が充填されます。 負の数値に関しては、記号は 16 進法 `Dx` に変換されます。 数値の小数点は無視されます。 小数点の前および後のすべての桁は、1 つの整数値として扱われます。
3. （**N**、**P**、**I**）からバイナリ（**1**～**4** バイト）
   数値はバイナリ（4 バイト）に変換されます。 数値の小数点は無視され、小数点の前後の値は 1 つの整数値として扱われます。 変換後のバイナリ値は、符号の値に応じて、正数または 2 の補数になります。
4. バイナリ（**1**～**4** バイト）から数値
   変換された値は右詰めで数値に割り当てられます。つまり、先行ゼロを含みます 長さが 1～3 バイトのバイナリ値は、常に正の符号を使用しているとみなされます。 長さが 4 バイトのバイナリ値は、最も左側のビットによって数値の符号が決まります（1 = 負数、0 = 正数）。受け取り側の数値の小数点は無視されます。 小数点の前および後のすべての桁は、1 つの整数値として扱われます。

5. バイナリからバイナリ
   値は右から左に 1 バイトずつ転送されます。 受け取りフィールドには、先頭にバイナリゼロが挿入されます。
6. バイナリ（**5** バイト以上）から英数字
   値は左から右に 1 バイトずつ移動します。 定義されている長さおよび指定されているバイト数に応じて、変換後の値は切り捨てられるか、または末尾に空白文字が追加されます。
7. 日付（**D**）から時刻（**T**）
   日付を時刻に転送する場合、時刻は 00:00:00:0 として変換されます。
8. 時刻（**T**）から日付（**D**）
   時刻を日付に転送する場合、時刻情報は切り捨てられ、日付情報のみが残されます。
9. **L**、**D**、**T**、**F** から **A**
   値は表示形式に変換され、左詰めで割り当てられます。
10. **F**
    短い英数字フィールドまたは Unicode フィールドに F を割り当てる場合、必要に応じて仮数が縮小されます。
11. **Unicode** から英数字
    Unicode 値は、International Components for Unicode（ICU）ライブラリを使用して、デフォルトのコードページ（システム変数 `*CODEPAGE` の値）に基づいた英数字に変換されます。 定義された長さおよび指定されたバイト数に応じて、結果が切り捨てられるか、結果の末尾に空白文字が追加されます。 Unicode 値の文字列がデフォルトのコードページに定義されていない場合、プロファイル／セッションパラメータ `CPCVERR` の設定に応じて、ランタイムエラーが出力されるか、または置換文字に置き換えられます。
12. **Unicode** からバイナリ
    値は左から右にコード単位で転送されます。 定義された長さおよび指定されたバイト数に応じて、結果が切り捨てられるか、結果の末尾に空白文字が追加されます。 受け取り側のバイナリフィールドの長さは偶数である必要があります。
13. 英数字から **Unicode**
    英数字値は、International Components for Unicode（ICU）ライブラリを使用して、デフォルトのコードページから Unicode 値に変換されます。 定義された長さおよび指定されたコード単位数に応じて、結果が切り捨てられるか、結果の末尾に空白文字が追加されます。
14. （**N**、**P**、**I**）およびバイナリ（長さ **1**～**4**）から **Unicode**
    値はアンパック形式に変換され、その値から先行ゼロを除去した英数字値が取得されます。 負の数値に関しては、記号は 16 進法 `Dx` に変換されます。 数値の小数点は無視されます。 小数点の前および後のすべての桁は、1 つの整数値として扱われます。 変換後の値が、英数字から Unicode に変換されます。 定義された長さおよび指定されたコード単位数に応じて、結果が切り捨てられるか、結果の末尾に空白文字が追加されます。
15. バイナリ（**5** バイト以上）から **Unicode**
    値は左から右に 1 バイトずつ移動します。 定義されている長さおよび指定されているバイト数に応じて、変換後の値は切り捨てられるか、または末尾に空白文字が追加されます。 送出側のバイナリフィールドの長さは偶数である必要があります。

16. **L**、**D**、**T**、**F** から **U**
    値は英数字の表示形式に変換されます。 変換後の値が、英数字から Unicode に変換され、左詰めで割り当てられます。

ソースとターゲットのフォーマットが同じ場合、定義されている長さと、指定されているバイト数（フォーマット A および B）またはコード単位（フォーマット U）に応じて、変換後の値は切り捨てられるか、あるいは末尾の空白（フォーマット A および U）または先頭のバイナリゼロ（フォーマット B）が追加されます。

「*ダイナミック変数の使用*」も参照してください。

フィールドの切り捨てと切り上げ

フィールドの切り捨ておよび切り上げには、以下のルールが適用されます。

- 数値フィールドの上位の切り捨ては、切り捨てる桁が先行ゼロの場合にのみ実行できます。 小数点以下の桁は、小数点が明示的に指定されているかどうかにかかわらず、切り捨てられます。
- 英数字フィールドの末尾の空白は切り捨てられます。
- オプション `ROUNDED` を指定すると、切り捨てられる値の一番上の桁が 5 以上の場合、結果の一番下の桁に切り上げられます。 除算の結果の精度については、「*算術演算結果の精度*」も参照してください。

算術演算結果のフォーマットと長さ

次の表は、算術演算の結果のフォーマットと長さを示しています。

	I1	I2	I4	N または P	F4	F8
I1	I1	I2	I4	P*	F4	F8
I2	I2	I2	I4	P*	F4	F8
I4	I4	I4	I4	P*	F4	F8
N または P	P*	P*	P*	P*	F4	F8
F4	F4	F4	F4	F4	F4	F8
F8	F8	F8	F8	F8	F8	F8

メインフレームコンピュータでは、算術演算結果の精度を向上させるため、フォーマット／長さ F4 の代わりに F8 が使用されます。

P* は、「*算術演算結果の精度*」に示されているように、各オペランドの整数の長さと精度に基づいて個別に決定されます。

フォーマット I に適用される、10 進数での桁数と有効な値を以下に示します。

フォーマット／長さ	10 進数での桁数	有効値
I1	3	-128～127
I2	5	-32,768～32,767
I4	10	-2,147,483,648～2,147,483,647

浮動小数点数を使用した算術演算

以下では次のトピックについて説明します。

- 全般的な考慮事項
- 浮動小数点数の精度
- 浮動小数点表現への変換
- プラットフォーム依存

全般的な考慮事項

浮動小数点数（フォーマット F）は整数（フォーマット I）と同様に 2 の累乗の和として表されますが、アンパック型およびパック型の数値（フォーマット N および P）は 10 の累乗の和として表されます。

アンパック型またはパック型の数値では、小数点の位置は固定です。一方、浮動小数点数では、小数点の位置は（その名が示すとおり）「浮動」 です。つまり、小数点の位置は固定されず、実際の値によって変わります。

浮動小数点数は、正弦関数や対数関数などの三角関数や算術関数の計算に不可欠です。

浮動小数点数の精度

浮動小数点数の性質から、精度には以下の制限があります。

- フォーマット／長さが F4 の変数の場合、精度は約 7 桁に制限されます。
- フォーマット／長さが F8 の変数の場合、精度は約 16 桁に制限されます。

これ以上大きな桁を持つ値は、浮動小数点数では正確に表せません。 小数点の前後に何桁あるかに関係なく、浮動小数点数で対応できるのはそれぞれ先頭の 7 桁または 16 桁のみです。

整数値は、絶対値が $2^{24} - 1$ を超えていなければ、フォーマット／長さが F4 の変数で正確に表すことができます。

浮動小数点表現への変換

英数字、アンパック型数値、またはパック型数値を浮動小数点フォーマットに変換する場合（割り当て操作など）、表現を変更、つまり、10 の累乗の和から 2 の累乗の和に変換する必要があります。

したがって、2の累乗の有限和で表現できる数値のみが正確に表されます。それ以外の数値は近似値としてのみ表されます。

例：

以下の数値は、正確な浮動小数点表現です。

$$1.25 = 2^{0} + 2^{-2}$$

以下の数値は、正確に表現できない浮動小数点表現です。

$$1.2 = 2^{0} + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-8} + 2^{-11} + 2^{-12} + \ldots$$

このように、英数字、アンパック型数値、またはパック型数値から浮動小数点値への変換、およびその逆の変換では、小規模なエラーが発生する場合があります。

注意: 整数、アンパック型、またはパック型の算術演算の結果（「*算術演算結果のフォーマットと長さ*」を参照）を浮動小数点表現に変換する必要がある場合、精度を向上させるために、最初から浮動小数点フォーマットで算術演算を実行することをお勧めします。

例：

```
#F1 (F8) := 1     / 12  /* Result is +8.333330000000000E-02
#F2 (F8) := 1.0E0 / 12  /* Result is +8.333333333333333E-02
```

プラットフォーム依存

ハードウェアアーキテクチャが異なるため、浮動小数点数の表現はプラットフォームによって変わります。浮動小数点演算を行う同じアプリケーションを別のプラットフォームで実行すると、結果がわずかに異なるのはこのためです。 それぞれの表現によって、浮動小数点変数の有効な値の範囲も決まります。（おおよその）有効な値の範囲は、次のとおりです。F4 および F8 の変数に対し、$\pm 5.4 * 10^{-79}$～$\pm 7.2 * 10^{75}$

注意: 電卓で使用されている表現も、コンピュータで使用されている表現と異なっている可能性があります。したがって、同じ計算をしても違う結果が出る可能性があります。

日付および時刻を使用した算術演算

フォーマットD（日付）およびT（時刻）は、加算、減算、乗算、および除算でのみ使用できます。 乗算および除算は、加算および減算の中間結果にのみ使用できます。

日付／時刻値は、相互に加算／減算できます。また、整数値（小数なし）を日付／時刻に加算／減算することもできます。 これらの整数値は、フォーマット N、P、I、D、または T のフィールドに格納できます。

加算／減算の中間結果は、その後の演算で被乗数／被除数として使用することもできます。

日付値に加算／減算する整数値は、日数とみなされます。 時刻値に加算／減算する整数値は、1/10 秒とみなされます。

日付と時刻を使用した算術演算には、以下のようなNaturalの内部処理に基づく制限が適用されます。

Natural では内部的に、日付／時刻変数を使用した算術演算は、以下のように処理されます。

```
COMPUTE result-field = operand1 +/- operand2
```

上のステートメントは次のように解決されます。

1. *intermediate-result = operand1 +/- operand2*
2. *result-field = intermediate-result*

つまり、Natural は、最初の手順で加算／減算の結果を計算し、2 番目の手順でその結果を結果フィールドに割り当てます。

さらに複雑な算術演算も、次のように、同じパターンに従って解決されます。

```
COMPUTE result-field = operand1 +/- operand2 +/- operand3 +/- operand4
```

上のステートメントは次のように解決されます。

1. *intermediate-result1 = operand1 +/- operand2*
2. *intermediate-result2 = intermediate-result1 +/- operand3*
3. *intermediate-result3 = intermediate-result2 +/- operand4*
4. *result-field = intermediate-result3*

乗算および除算も、加算および減算と同様に解決されます。

このような *intermediate-result* の内部フォーマットは、以下の表に示すとおり、オペランドのフォーマットに依存します。

加算

以下の表は、加算（*intermediate-result* = *operand1* + *operand2*）の中間結果のフォーマットを示しています。

operand1 のフォーマット	*operand2* のフォーマット	*intermediate-result* のフォーマット
D	D	Di
D	T	T
D	Di、Ti、N、P、I	D
T	D、T、Di、Ti、N、P、I	T
Di、Ti、N、P、I	D	D
Di、Ti、N、P、I	T	T
Di、N、P、I	Di	Di
Ti、N、P、I	Ti	Ti
Di	Ti、N、P、I	Di
Ti	Di、N、P、I	Ti

減算

以下の表は、減算（*intermediate-result* = *operand1* - *operand2*）の中間結果のフォーマットを示しています。

operand1 のフォーマット	*operand2* のフォーマット	*intermediate-result* のフォーマット
D	D	Di
D	T	Ti
D	Di、Ti、N、P、I	D
T	D、T	Ti
T	Di、Ti、N、P、I	T
Di、N、P、I	D	Di
Di、N、P、I	T	Ti
Di	Di、Ti、N、P、I	Di
Ti	D、T、Di、Ti、N、P、I	Ti
N、P、I	Di、Ti	P12

乗算または除算

以下の表は、乗算（*intermediate-result* = *operand1* * *operand2*）または除算（*intermediate-result* = *operand1* / *operand2*）の中間結果のフォーマットを示しています

operand1 のフォーマット	*operand2* のフォーマット	*intermediate-result* のフォーマット
D	D、Di、Ti、N、P、I	Di
D	T	Ti
T	D、T、Di、Ti、N、P、I	Ti
Di	T	Ti
Di	D、Di、Ti、N、P、I	Di
Ti	D	Di
Ti	Di、T、Ti、N、P、I	Ti
N、P、I	D、Di	Di
N、P、I	T、Ti	Ti

内部割り当て

Di は内部日付フォーマットの値です。Ti は内部時刻フォーマットの値です。これらの値は、その後の日付／時刻の算術演算に使用できますが、フォーマット D の結果フィールドに割り当てることはできません（以下の割り当て表を参照）。

内部フォーマット Di または Ti の中間結果をその後の加算／減算／乗算／除算でオペランドとして使用する複雑な算術演算では、これらのフォーマットはそれぞれ D または T とみなされます。

以下の表は、どの中間結果をどの結果フィールドに内部的に割り当て（*result-field* = *intermediate-result*）できるのかを示しています。

result-field のフォーマット	*intermediate-result* のフォーマット	割り当て可能
D	D、T	○
D	Di、Ti、N、P、I	×
T	D、T、Di、Ti、N、P、I	○
N、P、I	D、T、Di、Ti、N、P、I	○

フォーマット D または T の結果フィールドに、負数を格納することはできません。

例 **1** および **2**（無効）：

```
COMPUTE DATE1 (D) = DATE2 (D) + DATE3 (D)
COMPUTE DATE1 (D) = DATE2 (D) - DATE3 (D)
```

加算／減算の中間結果はフォーマット D*i* ですが、フォーマット D*i* の値をフォーマット D の結果フィールドに割り当てることはできないため、これらの演算は実行できません。

例 **3** および **4**（無効）：

```
COMPUTE DATE1 (D) = TIME2 (T) - TIME3 (T)
COMPUTE DATE1 (D) = DATE2 (D) - TIME3 (T)
```

加算／減算の中間結果はフォーマット T*i* ですが、フォーマット T*i* の値をフォーマット D の結果フィールドに割り当てることはできないため、これらの演算は実行できません。

例 **5**（有効）：

```
COMPUTE DATE1 (D) = DATE2 (D) - DATE3 (D) + TIME3 (T)
```

この演算は可能です。まず、DATE3 が DATE2 から減算され、フォーマット D*i* の中間結果が生成されます。次に、この中間結果が TIME3 に加算され、フォーマット T の中間結果が生成されます。最後に、この 2 番目の中間結果が結果フィールド DATE1 に割り当てられます。

例 **6** および **7**（無効）：

```
COMPUTE DATE1 (D) = DATE2 (D) + DATE3 (D) * 2
COMPUTE TIME1 (T) = TIME2 (T) - TIME3 (T) / 3
```

中間結果ではなく日付／時刻フィールドを使用して乗算／除算を実行しようとしているため、これらの演算は実行できません。

例 **8**（有効）：

```
COMPUTE DATE1 (D) = DATE2 (D) + (DATE3(D) - DATE4 (D)) * 2
```

この演算は可能です。まず、DATE4 が DATE3 から減算され、フォーマット Di の中間結果が生成されます。次に、この中間結果が 2 で乗算され、フォーマット Di の中間結果が生成されます。この中間結果が DATE2 に加算され、フォーマット D の中間結果が生成されます。最後に、この 3 番目の中間結果が結果フィールド DATE1 に割り当てられます。

フォーマット T の値をフォーマット D のフィールドに割り当てる場合、時刻値に有効な日付コンポーネントが含まれていることを確認する必要があります。

フォーマット表現の混在に対するパフォーマンスの考慮事項

算術演算を実行する場合、フィールドフォーマットの選択はパフォーマンスに大きく影響します。

業務演算では、フォーマット P（パック型数値）のフィールドのみを使用します。 すべてのオペランドの桁数は、できる限り一致させます。

科学演算では、可能であれば、フォーマット F（浮動小数点）のフィールドを使用します。

数値（N、P）と浮動小数点（F）のフォーマットが混在する式では、浮動小数点フォーマットへの変換が実行されます。 この変換は CPU にとってかなりの負荷となります。 したがって、算術演算ではフォーマット表現を混在させないことをお勧めします。

算術演算結果の精度

演算	整数部の桁数	小数部の桁数
加算／減算	Fi + 1 または Si + 1（どちらか大きい方）	Fd または Sd（どちらか大きい方）
乗算	Fi + Si + 2	Fd + Sd（最大 7）
除算	Fi + Sd	（下記参照）
累乗	15 - Fd（以下の「*例外*」を参照）	Fd
平方根	Fi	Fd

各項目の意味を次に示します。

F	第 1 オペランド
S	第 2 オペランド
R	結果
i	整数部の桁数
d	小数部の桁数

例外：

指数に1桁以上の小数部がある場合、累乗は内部的に浮動小数点フォーマットで行われ、その結果もまた浮動小数点フォーマットになります。 詳細については、「*浮動小数点数を使用した算術演算*」を参照してください。

除算の結果に対する小数部の桁数

除算の結果の精度は、結果フィールドを使用できるかどうかによって変わります。

- 結果フィールドを使用できる場合、精度は Rd または Fd（どちらか大きい方）になります*。
- 結果フィールドを使用できない場合、精度は Fd または Sd（どちらか大きい方）になります*。

* ROUNDED オプションを使用すると、実際に結果を四捨五入する前に、結果の精度が内部的に 1 桁増やされます。

結果フィールドは、COMPUTE ステートメント、DIVIDE ステートメント、および比較演算子の後に除算が使用されている論理条件（IF #A = #B / #C THEN ... など）では使用できる、または使用できるとみなされます。

結果フィールドは、比較演算子の前に除算が使用されている論理条件（IF #B / #C = #A THEN ... など）では使用できない、または使用できないとみなされます。

例外：

被除数と除数がともに整数フォーマットで、少なくともその1つが変数の場合、結果フィールドの精度および ROUNDED オプションが使用されているかどうかに関係なく、除算の結果は常に整数フォーマットになります。

算術式の結果の精度

算術式、例えば #A / (#B * #C) + #D * (#E - #F + #G) の精度は、処理順序に従って算術演算の結果を評価することにより導き出されます。 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。

算術演算のエラー条件

加算、減算、乗算、および除算では、結果の総桁数（整数部と小数部）が 31 を超えるとエラーになります。

累乗では、以下の条件のいずれかに該当するとエラーが発生します。

- 基数がパック型で、結果が 16 桁を超えたか、または任意の中間結果が 15 桁を超えた場合
- 基数が浮動小数点で、結果が概算で次の値を超えた場合：$7 * 10^{75}$

配列の処理

通常、次の規則が適用されます。

- スカラ演算はすべて、単一のオカレンスを構成する配列要素に適用されます。
- 定数値を使用して変数を定義すると（#FIELD (I2) CONSTANT <8> など）、コンパイル時に値が変数に割り当てられ、その変数は定数として処理されます。 したがって、このような変数を配列の添字として使用すると、その次元は*特定*の数のオカレンスを持つことになります。
- 割り当て／比較操作で次元数の異なる2つの配列を使用する場合、次元数の少ない配列の「足りない」 次元は (1:1) とみなされます。

  例：#ARRAY1 (1:2) を #ARRAY2 (1:2,1:2) に割り当てる場合、#ARRAY1 は #ARRAY1 (1:1,1:2) とみなされます。

以下では次のトピックについて説明します。

配列の次元の定義

1 次元、2 次元、および 3 次元の配列は、以下のように定義します。

次元数	プロパティ
3	#a3(第 3 次元, 第 2 次元, 第 1 次元)
2	#a2(第 2 次元, 第 1 次元)
1	#a1(第 1 次元)

配列の割り当て操作

配列の範囲を別の配列の範囲に割り当てる場合、割り当ては要素ごとに実行されます。

例：

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:5) INIT <10,20,30,40,50>
END-DEFINE
*
MOVE #ARRAY(2:4) TO #ARRAY(3:5)
/* is identical to
/* MOVE #ARRAY(2) TO #ARRAY(3)
/* MOVE #ARRAY(3) TO #ARRAY(4)
/* MOVE #ARRAY(4) TO #ARRAY(5)
/*
/* #ARRAY contains  10,20,20,20,20
```

単一のオカレンスを配列の範囲に割り当てる場合、範囲の各要素に単一のオカレンスの値が割り当てられます。 算術関数を使用する場合、範囲の各要素に関数の結果が割り当てられます。

割り当て操作を実行する前に、配列の各次元を相互に比較して、以下にリストされている条件に該当するかどうかをチェックします。 次元は個別に比較されます。つまり、配列の第 1 次元はもう 1 つの配列の第 1 次元と、配列の第 2 次元はもう 1 つの配列の第 2 次元と、そして配列の第 3 次元はもう 1 つの配列の第 3 次元と比較されます。

ある配列から別の配列への値の割り当ては、以下の条件のいずれかに該当する場合にのみ実行できます。

- 比較する次元のオカレンス数が一致している場合
- 比較する次元の両方のオカレンス数が無限の場合
- 別の次元に割り当てられる次元が単一のオカレンスで構成されている場合

例 - 配列の割り当て：

以下のプログラムは、実行可能な配列の割り当て操作を示しています。

```
DEFINE DATA LOCAL
1 A1   (N1/1:8)
1 B1   (N1/1:8)
1 A2   (N1/1:8,1:8)
1 B2   (N1/1:8,1:8)
1 A3   (N1/1:8,1:8,1:8)
1 I    (I2)          INIT <4>
1 J    (I2)          INIT <8>
1 K    (I2)          CONST <8>
END-DEFINE
*
COMPUTE A1(1:3) = B1(6:8)                                    /* allowed
COMPUTE A1(1:I) = B1(1:I)                                    /* allowed
COMPUTE A1(*)   = B1(1:8)                                    /* allowed
COMPUTE A1(2:3) = B1(I:I+1)                                  /* allowed
```

```
 COMPUTE A1(1)   = B1(I)                                          /* allowed
 COMPUTE A1(1:I) = B1(3)                                          /* allowed
 COMPUTE A1(I:J) = B1(I+2)                                        /* allowed
 COMPUTE A1(1:I) = B1(5:J)                                        /* allowed
 COMPUTE A1(1:I) = B1(2)                                          /* allowed
 COMPUTE A1(1:2) = B1(1:J)                                        /* NOT ALLOWED
(NAT0631)
 COMPUTE A1(*)   = B1(1:J)                                        /* NOT ALLOWED
(NAT0631)
 COMPUTE A1(*)   = B1(1:K)                                        /* allowed
 COMPUTE A1(1:J) = B1(1:K)                                        /* NOT ALLOWED
(NAT0631)
 *
 COMPUTE A1(*)       = B2(1,*)                                    /* allowed
 COMPUTE A1(1:3)     = B2(1,I:I+2)                                /* allowed
 COMPUTE A1(1:3)     = B2(1:3,1)                                  /* NOT ALLOWED
(NAT0631)
 *
 COMPUTE A2(1,1:3)   = B1(6:8)                                    /* allowed
 COMPUTE A2(*,1:I)   = B1(5:J)                                    /* allowed
 COMPUTE A2(*,1)     = B1(*)                                      /* NOT ALLOWED
(NAT0631)
 COMPUTE A2(1:I,1)   = B1(1:J)                                    /* NOT ALLOWED
(NAT0631)
 COMPUTE A2(1:I,1:J) = B1(1:J)                                    /* allowed
 *
 COMPUTE A2(1,I)     = B2(1,1)                                    /* allowed
 COMPUTE A2(1:I,1)   = B2(1:I,2)                                  /* allowed
 COMPUTE A2(1:2,1:8) = B2(I:I+1,*)                                /* allowed
 *
 COMPUTE A3(1,1,1:I)   = B1(1)                                    /* allowed
 COMPUTE A3(1,1,1:J)   = B1(*)                                    /* NOT ALLOWED
(NAT0631)
  COMPUTE A3(1,1,1:I)   = B1(1:I)                                 /* allowed
 COMPUTE A3(1,1:2,1:I) = B2(1,1:I)                                /* allowed
 COMPUTE A3(1,1,1:I)   = B2(1:2,1:I)                              /* NOT ALLOWED
(NAT0631)
 END
```

配列の比較操作

一般的に、複数の次元を持つ配列を比較する場合、各次元は個別に処理されます。つまり、配列の第 1 次元はもう 1 つの配列の第 1 次元と、配列の第 2 次元はもう 1 つの配列の第 2 次元と、そして配列の第 3 次元はもう 1 つの配列の第 3 次元と比較されます。

2 つの配列の次元の比較は、以下の条件のいずれかに該当する場合にのみ実行できます。

- 比較する配列の次元のオカレンス数が同じ場合
- 比較する配列の次元のオカレンス数が無限の場合
- どちらかの配列の全次元が単一のオカレンスの場合

例 - 配列の比較：

以下のプログラムは、実行可能な配列の比較操作を示しています。

```
DEFINE DATA LOCAL
1 A3  (N1/1:8,1:8,1:8)
1 A2  (N1/1:8,1:8

 1 A1  (N1/1:8)
1 I   (I2)   INIT <4>
1 J   (I2)   INIT <8>
1 K   (I2)   CONST <8>
END-DEFINE
*
IF A2(1,1)     = A1(1)              THEN IGNORE END-IF /* allowed
IF A2(1,1)     = A1(I)              THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(1)              THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(I)              THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(*)              THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(I -3:I+4)       THEN IGNORE END-IF /* allowed
IF A2(1,5:J)   = A1(1:I)            THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(1:I)            THEN IGNORE END-IF /* NOT ALLOWED(NAT0629)
IF A2(1,*)     = A1(1:K)            THEN IGNORE END-IF /* allowed
*
IF A2(1,1)     = A2(1,1)            THEN IGNORE END-IF /* allowed
IF A2(1,1)     = A2(1,I)            THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A2(1,1:8)          THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A2(I,I -3:I+4)     THEN IGNORE END-IF /* allowed
IF A2(1,1:I)   = A2(1,I+1:J)        THEN IGNORE END-IF /* allowed
IF A2(1,1:I)   = A2(1,I:I+1)        THEN IGNORE END-IF /* NOT ALLOWED(NAT0629)
IF A2(*,1)     = A2(1,*)            THEN IGNORE END-IF /* NOT ALLOWED(NAT0629)
IF A2(1,1:I)   = A1(2,1:K)          THEN IGNORE END-IF /* NOT ALLOWED(NAT0629)
*
IF A3(1,1,*)   = A2(1,*)            THEN IGNORE END-IF /* allowed
IF A3(1,1,*)   = A2(1,I -3:I+4)     THEN IGNORE END-IF /* allowed
IF A3(1,*,I:J) = A2(*,1:I+1)        THEN IGNORE END-IF /* allowed
IF A3(1,*,I:J) = A2(*,I:J)          THEN IGNORE END-IF /* allowed
END
```

2 つの配列の範囲を比較する場合、以下の 2 つの表現では結果が異なることに注意してください。

```
#ARRAY1(*) NOT EQUAL #ARRAY2(*)
NOT #ARRAY1(*) = #ARRAY2(*)
```

例：

- 条件 **A**：

  ```
  IF #ARRAY1(1:2) NOT EQUAL #ARRAY2(1:2)
  ```

  これは、以下と等しくなります。

  ```
  IF (#ARRAY1(1) NOT EQUAL #ARRAY2(1)) AND (#ARRAY1(2) NOT EQUAL #ARRAY2(2))
  ```

  したがって、#ARRAY1 の最初のオカレンスと #ARRAY2 の最初のオカレンスが異なり、かつ、#ARRAY1 の 2 番目のオカレンスと #ARRAY2 の 2 番目のオカレンスが異なっている場合に、条件 A は真になります。

- 条件 **B**：

  ```
  IF NOT #ARRAY1(1:2) = #ARRAY2(1:2)
  ```

  これは、以下と等しくなります。

  ```
  IF NOT (#ARRAY1(1)= #ARRAY2(1) AND #ARRAY1(2) = #ARRAY2(2))
  ```

  また、以下とも等しくなります。

  ```
  IF (#ARRAY1(1) NOT EQUAL #ARRAY2(1)) OR (#ARRAY1(2) NOT EQUAL #ARRAY2(2))
  ```

  したがって、#ARRAY1 の最初のオカレンスと #ARRAY2 の最初のオカレンスが異なっているか、または、#ARRAY1 の 2 番目のオカレンスと #ARRAY2 の 2 番目のオカレンスが異なっている場合に、条件 B は真になります。

配列での算術演算

配列を使用する算術演算の一般的なルールは、対応する次元のオカレンス数が一致している必要があるということです。

以下の例は、このルールに従っています。

```
#c(2:3,2:4) := #a(3:4,1:3) + #b(3:5)
```

つまり、次のようになります。

配列	次元	オカレンス数	範囲
#c	第 2 次元	2	2:3
#c	第 1 次元	3	2:4
#a	第 2 次元	2	3:4
#a	第 1 次元	3	1:3
#b	第 1 次元	3	3:5

演算は要素ごとに実行されます。

注意: 次元数の異なる算術演算を実行することもできます。

上記の例では、以下の演算が実行されます。

```
#c(2,2) := #a(3,1) + #b(3)

#c(2,3) := #a(3,2) + #b(4)

#c(2,4) := #a(3,3) + #b(5)

#c(3,2) := #a(4,1) + #b(3)

#c(3,3) := #a(4,2) + #b(4)

#c(3,4) := #a(4,3) + #b(5)
```

以下のリストは、（COMPUTE、ADD、および MULTIPLY の各ステートメントにおける）算術演算で配列の範囲がどのように使用されるかを例示したものです。 例 1～4 では、対応する次元のオカレンス数が一致している必要があります。

1. *range* + *range* = *range*

   加算は要素ごとに実行されます。

2. *range* * *range* = *range*

   乗算は要素ごとに実行されます。

3. *scalar* + *range* = *range*

   スカラが範囲の各要素に加算されます。

4. *range* * *scalar* = *range*

   範囲の各要素がスカラで乗算されます。

5. *range* + *scalar* = *scalar*

   範囲の各要素がスカラに加算され、その結果がスカラに割り当てられます。

6. *scalar* * *range* = *scalar2*

   スカラが配列の各要素で乗算され、その結果が *scalar2* に割り当てられます。

例 1〜4 に示されているように、算術演算では中間結果が生成されないため、計算結果のフォーマット（「*算術演算結果のフォーマットと長さ*」を参照）は結果オペランドのフォーマットと同じである必要があります（フォーマット P および N は同じとみなされます）。

例：

```
DEFINE DATA LOCAL
1 #ARRAYI4(I4/1:5)
1 #ARRAYP5(P5/1:5)
END-DEFINE
*
#ARRAYI4(*) := #ARRAYP5(*) + 1  /* NOT ALLOWED(NAT0294)
```

上記の例に示されているように、算術演算では中間結果が生成されないため、重複する添字範囲は要素ごとに計算されます。

例：

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:5) INIT <10,20,30,40,50>
END-DEFINE
*
#ARRAY(3:5) := #ARRAY(2:4) + 1
/* is identical to
/* #ARRAY(3) := #ARRAY(2) + 1
/* #ARRAY(4) := #ARRAY(3) + 1
/* #ARRAY(5) := #ARRAY(4) + 1
/*
/* #ARRAY contains  10,20,21,22,23
```

53 コンパイルのポイント

以下はコンパイルのオプションおよびパラメータの概要です。コンパイルのオプションおよびパラメータは、作成したソースコードがコンパイラによってどうチェックされるか、およびNaturalランタイムシステムが解釈および実行できるNatural内部オブジェクトコードにどう変換されるかに影響します。

Naturalコンパイラおよびソースコードをコンパイルするためのさまざまなシステムコマンドの詳細については、『*Natural* システムアーキテクチャ』ドキュメントの「*Natural* コンパイラ」を参照してください。

コンパイラのオプションとパラメータ

コンパイルオプションは次のレベルで指定できます。

1. 静的

   Natural パラメータモジュールで次を指定

   - パラメータマクロ `NTCMPO`
   - プロファイルパラメータ `FS`、`XREF`

2. 動的

   プロファイルパラメータで次を指定

   - `CMPO`
   - `FS`、`XREF`

3. **Natural** セッション内

   次を指定

   - `COMPOPT` システムコマンドオプション
   - `GLOBALS` システムコマンド（セッションパラメータ `FS`）

4. 現在の **Natural** プログラミングオブジェクトに対して

   次を指定

   - 1 つまたは複数の `OPTIONS` ステートメント

     （`COMPOPT` システムコマンドと同じオプション、および加えてNatural Optimizer Compilerオプションを指定）

   - `SET GLOBALS` ステートメント（セッションパラメータ `LS`、`PS`、`ZP` のみ）

コンパイラに影響するその他のパラメータ

次のプロファイルパラメータまたはセッションパラメータはコンパイル時に考慮され、ソースコードに異なる設定が含まれている場合にエラーメッセージが表示されることがあります。

パラメータ名	簡単な説明	コメント
CFICU	Unicode およびコードページのサポート	I/O ステートメントでの Unicode 定数の出力（WRITE U'ABC' など）は、CFICU=ON の場合のみ可能です。
CP	デフォルトのコードページ名	ソースのコードページを指定します。
DC	小数点表記の文字	フォーマット N または P と定数によるフィールドを定義する際に整数部と小数部の区切りに使用する文字です。
FS	フォーマット指定	それまでに定義されていない変数の使用を許可または拒否します。
ID	INPUT 区切り文字	例えば、INIT 節で値を区切るために使用します。
LS	Line Size	レポートの出力行のデフォルトサイズを指定します（0～31）。この設定は、ソースコードで FORMAT ステートメントを明示的に指定することで上書きできます。
PS	Page Size	単一の I/O ステートメント（WRITE など）で出力可能な最大行数です。 基本設定に違反するとエラーになります。 この設定は、ソースコードで FORMAT ステートメントを明示的に指定することで上書きできます。
SF	フィールド間の空白	DISPLAY ステートメントを使用して作成される Natural レポート上で、列のフィールド設定間に挿入されるデフォルトのスペース数を指定します。この設定は、ソースコードで FORMAT ステートメントを明示的に指定することで上書きできます。
SOSI	ダブルバイト文字セットのシフトアウト／シフトインコード	ダブルバイト文字セット（DBCS）設定を示すシフト文字を指定します。
ZP	ゼロ出力	値がすべて零の数値定数について、空白またはゼロを出力します。 この設定は、ソースコードで FORMAT ステートメントを明示的に指定することで上書きできます。

54 インターネットおよびXMLアクセス用のステートメント

この章では、インターネットおよびXMLにアクセスするためのステートメントの機能概要、メインフレーム環境でこれらのステートメントを使用するための全般的な前提条件、制限事項、およびその他の参照情報について説明します。これらのステートメントを最大限に活用するには、通信規格に関する深い知識が必要です。

使用可能なステートメント

次の Natural ステートメントを使用して、インターネットおよび XML ドキュメントにアクセスできます。

- REQUEST DOCUMENT
- PARSE XML

REQUEST DOCUMENT

- 機能
- 技術的な実装
- 構文
- REQUEST DOCUMENT のプラットフォームサポート

機能

このステートメントを使用すると、ハイパーテキスト転送プロトコル（HTTP）、および z/OS の場合のみ Hypertext Transfer Protocol Secure（HTTPS）を使用して、与えられた Uniform Resource Identifier（URI）または Uniform Resource Locator（URL）を持つ Web 上のドキュメントにアクセスできます。URI および URL は、Web サイトのインターネットアドレスまたはイントラネットアドレスです。

REQUEST DOCUMENT では HTTP クライアントを Natural ステートメントレベルに実装します。これにより、アプリケーションはイントラネットまたはインターネット上の任意の HTTP サーバーにアクセスできます。 このステートメントにはオペランドのセットがあり、このオペランドを使用してユーザーアプリケーションの要件に沿った HTTP 要求を作成できます。 例えば、送信オペランドを使用すると、ユーザー定義 HTTP ヘッダー、フォームデータ、またはドキュメント全体を HTTP サーバーに送信できます。 受信オペランドを使用すると、サーバーからドキュメントを取得したり、サーバーから返された HTTP ヘッダーブロック全体を表示したり、専用のヘッダーの値を戻したりできます。バイナリフォーマットオペランドを使用すると、gif ファイルなどのバイナリオブジェクトを HTTP サーバーと交換できます。 基本認証用として、ユーザー ID やパスワードのオペランドを指定できます。 このオペランドの内容は、HTTP 標準に従って回線上を base64 エンコーディングを使用して送信されます。

Natural では次の要求メソッドがサポートされています。

- GET - ドキュメントと HTTP ヘッダーの取得
- HEAD - HTTP ヘッダーのみの取得

- POST - フォームデータの HTTP サーバーへの転送
- PUT - ファイルの HTTP サーバーへの転送

要求メソッドは、通常、実行される REQUEST DOCUMENT ステートメントにコーディングされているオペランドに基づいて、自動的に評価されます。 ただし、あらかじめ設定されている要求メソッドは、要求メソッドヘッダーをユーザーが明示的に指定することによって上書きできます。

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。送信データ／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は、REQUEST DOCUMENT ステートメントの DATA ALL 節／RETURN PAGE 節を使用して指定できます。

HTTP サーバーを持つ EBCDIC ベースのメインフレームは、ほとんどが UTF-8 または ISO-8859-1 でエンコードされたデータを使用しています。このようなメインフレームとのデータ交換を簡略化するため、このステートメントでは ENCODED 節を使用して、送受信ドキュメントデータを暗黙的または自動的に変換できます。

技術的な実装

REQUEST DOCUMENT ステートメントの実装は、主に次の 2 階層で構成されています。

- 独立したランタイム層。HTTP 処理、URL 解析、データ変換などの一切が実行されます。
- 環境依存ルーチンが Natural と HTTP サーバーとの TCP/IP 通信を処理する層。 この層は、z/OS、VSE、および VM/CMS の場合は LE（言語環境）ソケットに基づいて、Com-plete および Natural 開発サーバーの場合は SMARTS ソケットに基づいて、BS2000/OSD の場合は CRTE ソケットに基づいて、それぞれ実装されます。 CICS の場合は、適切なソケットライブラリがビルドプロセスに追加されます。

Natural for Mainframes では HTTP プロトコルバージョン 1.0 のみがサポートされています。つまり、サーバーへの固定接続は維持されません。 事実上、すべての企業のネットワークプロセスでは、クライアントからインターネットにアクセスするとき、プロキシサーバーを経由します。このため、Naturalは、プロキシサーバーおよびプロキシサーバーの処理対象となるポートに適した設定を使用して構成できるようになっています。 また、プロキシサーバーを経由せずに直接アクセスする、ローカルドメイン名の接尾辞（イントラネットサイト）も指定できます。「*適用可能な **Natural** パラメータの概要*」も参照してください。

プロキシサーバーは、クライアント（ユーザー）とインターネットの間に配置され、クライアントからの要求を受信して宛先サーバーに転送したり、返されたドキュメントをキャッシュしてクライアントに転送したりします。 プロキシサーバーの利点として、キャッシュ処理によってパフォーマンスが向上すること、およびセキュリティの問題の回避に役立つこと（ほとんどのプロキシサーバーがファイアウォールとしても機能する）を挙げることができます。

次の例は、REQUEST DOCUMENT ステートメントを使用して、外部に保管されているドキュメントにアクセスする方法を示しています。

```
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
```

構文

REQUEST DOCUMENT ステートメントの構文と詳細なアプリケーションヒントについては、『ステートメント』ドキュメントを参照してください。

REQUEST DOCUMENT のプラットフォームサポート

REQUEST DOCUMENT ステートメントは、次のメインフレームプラットフォームでサポートされます。

- **z/OS**：Batch、TSO、CICS、Com-plete、IMS/TM
- **z/VSE**：Batch、Com-plete、CICS
- **VM/CMS**
- **BS2000/OSD**：Batch、TIAM、*open*UTM *

* 後述の「*openUTM での XML 関連ステートメントのサポートに対する前提条件*」も参照してください。

また、このステートメントは Natural でサポートされているすべての OpenSystems プラットフォームで使用できます。

PARSE XML

- PARSE XML のプラットフォームサポート

機能

PARSE XML ステートメントを使用すると、Natural プログラム内で XML ドキュメントを解析できるようになります。

PARSE XML ステートメントではフル XML パーサを Natural に統合します。これにより、Natural アプリケーションで XML ドキュメントを解析できるようになり、コンテンツの処理が容易になります。 PARSE XML ステートメントでは処理ループを開き、解析プロセス中にリストのイベントのいずれかが発生するたびに、それぞれのドキュメントのパス、解析された要素の名前と値、およびいくつかのパーサステータスシステム変数を返します。

技術的な実装

XML ドキュメント解析の一般的な戦略またはモデルは次のとおりです。

- DOM（Document Object Model）。オブジェクト指向アプローチの一種です。
- SAX（Simple Access to XML）。ストリーム指向の解析メソッドです。

Natural for Mainframes での PARSE XML ステートメントの実装は SAX メソッドに基づいており、（オープンソース）SAX パーサ EXPAT のバージョン 2.0.0 のメインフレームポートを使用しています。

解析処理は、対象ドキュメントの UTF-16 エンコードイメージに対して内部的に行われます。つまり、配信されたドキュメントのエンコーディングが違う場合は、UTF-16 への変換が内部的に実行されてから解析が開始されます。 例えば、TP 環境のスレッドサイズを評価する場合は、Natural をインストールときにこのことを考慮する必要があります。

解析するドキュメントのエンコーディングは自動的に確認されます。

1. ドキュメントのエンコーディングを示す BOM（バイトオーダーマーク）が確認されます。
2. BOM が見つからない場合、ASCII、EBCDIC、または UTF-16（BE または LE、つまりビッグエンディアンまたはリトルエンディアン）が確認されます。
3. エンコーディングが EBCDIC または ASCII の場合は、エンコーディングの処理命令が検索されます。

   エンコーディングが不明の場合は、該当するエラーメッセージが発行され、解析プロセスは終了します。 内部的に、パーサは UTF-16BE で動作するため、解析されるドキュメントは必ずこのエンコーディングに変換されてから EXPAT パーサに渡されます。
4. エンコーディングが PI（処理命令）の場合は、次のデフォルトが適用されます。
   - ASCII の場合は、UTF-8 エンコーディングが想定されます。
   - EBCDIC の場合は、Natural デフォルトコードページ（システム変数 *CODEPAGE を参照）エンコーディングが想定されます。

解析プロセス自体は、2 段階で構成されます。

- 最初の段階では、コールバックエントリの整形式セットを通知するために、パーサが繰り返し呼び出されます。 その時点で解析中のドキュメントに、これらのエントリに関連する要素が出現するたび、これらのエントリがパーサによって入力されます。 例えば、開始タグが出現することは、対応するエントリへのコールバックをトリガするイベントとなります。 コールバックエントリは、解析プロセスの実行のためのNaturalランタイムロジックを公開します。
- 第2段階は、実際の解析プロセスです。解析するドキュメントを入力オペランドとして、パーサが呼び出されます。 この段階で各要素が解析され、各要素タイプに対応するコールバックルーチンが呼び出されます。次に、Naturalランタイムでは返された要素を処理し、返されるオペランドを更新し、それらのオペランドを処理するための解析ループに入ります。その後、パーサが再起動され、解析プロセスが続行されます。 解析プロセスは、ドキュメントの解析が完了したとき、および現在のドキュメントで XML 構文エラーが発生したとき、つまりドキュメントの形式が無効であった場合に完了します。

注意: 技術的な理由により、Natural for Mainframes ではネスト構造の解析ループはサポートされていません。

XML の空白文字と事前定義されたエンティティの処理

Natural バージョン 4.2.5 以降、解析対象文字列に空白文字または事前定義された XML エンティティが含まれていた場合に、文字データの解析でブレイクまたはループパスが発生することはありません。 バージョン 4.2.5 より前の Natural で問題になっていましたが、解決されました。 Natural バージョン 4.2.5 では、文字データの解析は Natural for Windows、UNIX、および Linux と互換性があります。

次のサンプルプログラムの出力は、バージョン 4.2.5 とそれより前のバージョンとの違いを示します。

```
DEFINE DATA
LOCAL
1 PAGE    (A)   DYNAMIC
1 #PATH   (A200)
1 #NAME   (A) DYNAMIC
1 #VALUE  (A40)
1 #CMX    (A) DYNAMIC
1 #CMP    (A) DYNAMIC
END-DEFINE
FORMAT PS=60 LS=80
COMPRESS ' A&lt;B ' H'0D0D' ' B&lt;C' INTO #CMX LEAVING NO
MOVE ALL  #CMX TO #CMP UNTIL 16
COMPRESS
'<?xml version="1.0" ?>'
'<character-data-sample>'
'<string_with_whitespace_and_predefined_entity>' #CMX
'</string_with_whitespace_and_predefined_entity>'
'</character-data-sample>'
```

```
 INTO PAGE LEAVING NO
PARSE XML PAGE INTO PATH #PATH NAME #NAME VALUE #VALUE
PRINT #PATH /   'NA='  #NAME / 'VA=' #VALUE
LOOP
END
```

Natural バージョン 4.2.5 より前でプログラムが実行された場合の出力：

```
Page      1                                                  08-11-04  14:39:51

character-data-sample
NA= character-data-sample
VA=
character-data-sample/string_with_whitespace_and_predefined_entity
NA= string_with_whitespace_and_predefined_entity
VA=
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA=  A
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= <
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= B
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= ?
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= ?
NA=
VA= B
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= ?
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= ?
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA=  B
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= <
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= C
character-data-sample/string_with_whitespace_and_predefined_entity//
NA= string_with_whitespace_and_predefined_entity
VA=
character-data-sample//
```

```
NA= character-data-sample
VA=
MORE
```

Natural バージョン 4.2.5（以降）でプログラムが実行された場合の出力：

```
Page      1                                                  08-11-04  13:41:34

character-data-sample
NA= character-data-sample
VA=
character-data-sample/string_with_whitespace_and_predefined_entity
NA= string_with_whitespace_and_predefined_entity
VA=
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA=  A<B?? B<C
character-data-sample/string_with_whitespace_and_predefined_entity//
NA= string_with_whitespace_and_predefined_entity
VA=
character-data-sample//
NA= character-data-sample
VA=


MORE
```

構文

PARSE XML ステートメントの構文と詳細なアプリケーションヒントについては、『ステートメント』ドキュメントを参照してください。

PARSE XML のプラットフォームサポート

PARSE XML ステートメントは、次のメインフレームプラットフォームでサポートされます。

- **z/OS**：Batch、TSO、CICS、Com-plete、IMS/TM *
- **z/VSE**：Batch、Com-plete、CICS
- **VM/CMS**
- **BS2000/OSD**：Batch、TIAM、*open*UTM **

* 後述の「*IMS/TM に関する制限事項*」を参照してください。

** 後述の「*openUTM での XML 関連ステートメントのサポートに対する前提条件*」も参照してください。

また、このステートメントは Natural でサポートされているすべての OpenSystems プラットフォームで使用できます。

全般的な前提条件

このセクションでは、Natural ステートメント REQUEST DOCUMENT および PARSE XML を使用する場合に適用される前提条件について説明します。

インストールの前提条件

Natural ステートメント REQUEST DOCUMENT および PARSE XML の使用を有効にするには、『インストール』ドキュメントの「*REQUEST DOCUMENT および PARSE XML ステートメントサポートのインストール*」で説明されているインストール手順を実行する必要があります。

REQUEST DOCUMENT および PARSE XML は、少なくとも内部的には必ずデータのエンコーディングを変換する必要があるため、Natural の稼働にはアクティブな ICU サポートが必要です。 このため、ICU ライブラリをインストールする必要があります。

REQUEST DOCUMENT または PARSE XML を使用する予定の場合は、次の前提条件を満たす必要があります。

- 実行環境で TCP/IP スタックが使用可能で有効になっていること
- インターネットアドレス解決要求（gehthostbyname 関数）を解決するため、実行環境で DNS（Domain Name System）サーバーまたは DNS サービスが使用可能であること
- Natural ドライバが LE 対応（IBM 環境の場合）または CRTE 対応（BS2000/OSD 環境）でインストールされていること
- Com-plete で HTTPS をサポートする場合は、APS バージョン 2.7.2 パッチレベル 16

プロファイル設定

適用可能な **Natural** パラメータの概要

次の表に、ステートメント REQUEST DOCUMENT および PARSE XML のサポートを有効化および無効化するか、またはサポートに影響を及ぼす、Natural プロファイルパラメータおよびセッションパラメータの概要を示します。

パラメータ	目的
XML	この Natural プロファイルパラメータまたは対応するパラメータマクロ NTXML とそのキーワードサブパラメータは、ステートメント REQUEST DOCUMENT および PARSE XML の有効化／無効化に使用されます。 また、NTXML や XML のキーワードサブパラメータを使用して設定できるさまざまなオプションがあります。この例として、ステートメント REQUEST DOCUMENT および PARSE XML のサポートの有効化／無効化の個別指定、デフォルトコードページの名前、（イントラネット）プロキシサーバーの URL、プロキシのポート番号、（イントラネット）SSL プロキシサーバーの URL とポート番号、アドレスを直接指定するローカルドメインの名前を挙げることができます。 XML プロファイルパラメータを使用する前提条件として、プロファイルパラメータ CFICU を ON に設定する必要があります。
CFICU	この Natural プロファイルパラメータまたは対応するパラメータマクロ NTCFICU とそのキーワードサブパラメータは、Unicode およびコードページサポートを有効にするために使用します。
CP	この Natural プロファイルパラメータでは、Natural データおよび Natural ソースのデフォルトのコードページを定義します。
CPCVERR	この Natural プロファイルおよびセッションパラメータでは、変換時に発生した変換エラーが Natural エラーにつながるかどうかを指定します。

これらのパラメータの詳細については、『*パラメータリファレンス*』ドキュメントの対応セクションを参照してください。

有効化／無効化

手順 54.1. ステートメント REQUEST DOCUMENT および PARSE XML のサポートを*現在*のセッションで有効にするには

1 両ステートメントをともに有効にするには、Natural プロファイルパラメータ XML を ON に設定します。 システムコマンド GLOBALS を使用するか、Natural の起動時にこのパラメータを動的に指定します。

または:

サポートを*個別に*設定するには、XML/NTXMLの該当するキーワードサブパラメータのみONに設定します。

RDOC - REQUEST DOCUMENTサポートの有効化

PARSE - PARSE XMLサポートの有効化

2 インストールプラットフォームがインターネットファイアウォールの背後にある場合、またはインターネットトラフィックがプロキシサーバー経由でルーティングされる場合は、プロキシおよびプロキシポートに関するXML/NTXMLのキーワードサブパラメータをそれに合わせて指定する必要があります。

手順 54.2. ステートメントREQUEST DOCUMENTおよびPARSE XMLのサポートをすべてのセッションについて有効にするには

■ 「*適用可能なNaturalパラメータの概要*」にリストされているパラメータやマクロをNaturalパラメータモジュールNATPARMに追加し、該当する値を設定するよう、システム管理者に依頼します。

手順 54.3. ステートメントREQUEST DOCUMENTおよびPARSE XMLのサポートを無効にするには

■ 両ステートメントをともに無効にするには、Naturalプロファイルパラメータ XMLまたはパラメータマクロ NTXMLを OFFに設定します。

または:

サポートを*個別に*無効にするには、XML/NTXMLの該当するキーワードサブパラメータのみOFFに設定します。

RDOC - REQUEST DOCUMENTサポートの無効化

PARSE - PARSE XMLサポートの無効化

詳細については、『*パラメータリファレンス*』ドキュメントの「*XML - PARSE XMLおよびREQUEST DOCUMENTステートメントの有効化*」も参照してください。

Unicodeサポート

手順 54.4. Unicodeサポートを有効にするには

■ プロファイルパラメータ CFICUを ONに設定します。

プロファイルパラメータ CFICUのキーワードサブパラメータに設定できるさまざまなオプションの詳細については、『*パラメータリファレンス*』ドキュメントの「*CFICU - Unicodeとコードページのサポート*」を参照してください。

『*Unicode とコードページのサポート*』ドキュメントの「*Natural プログラミング言語の Unicode とコードページのサポート*」の「ステートメント」に記載されている、PARSE XML および REQUEST DOCUMENT に関連する説明も参照してください。

z/OS 環境での REQUEST DOCUMENT ステートメントの HTTPS サポート

HTTPS の概要

HTTPS は Hypertext Transfer Protocol Secure の略で、HTTP と TCP/IP プロトコルスタックとの間の追加セキュリティ層です。

階層	プロトコル
アプリケーション層	HTTP（S）
セキュリティ層	TLS/SSL
トランスポート層	TCP
ネットワーク層	IP

HTTPS は、インターネット経由での安全なデータ通信を目的として、暗号化と通信パートナー認証を実現するために導入されました。

HTTPS URI スキームは、HTTP 通信がセキュリティ保護されることを示すために使用されます。データの暗号化には、SSL（Secure Socket Layer）プロトコルまたはその後継である TLS（Transport Layer Security）が使用されます。 そのため認証は、通信パートナーの身元を保証する証明書の交換によって行われます。

ただし、ほとんどの HTTPS 通信において、サーバーが一方的に証明書を使用してクライアントの認証を受けます。クライアント証明書を使用したクライアント認証はほとんど行われません。

SSL 通信はいくつかの段階を経て確立されます。

- まず、いわゆる SSL ハンドシェイクプロトコル（Client Hello、Server Hello）による通信パートナーの識別と認証が行われます。
- このハンドシェイクが終わると、非対称暗号化を使用した対称セッションキーの交換が行われます（秘密 - 公開キー処理）。 公開キーはサーバー証明書に不可欠な部分で、これ以降クライアントによって使用されます。

- ハンドシェイクとキー交換の実行後、暗号化されたペイロード要求メッセージが交換されます。 前の手順でネゴシエーションされた対称セッションキーは、メッセージの暗号化／復号化に使用されます。

HTTPS プロトコルでは、標準的な HTTP ポート番号とは違うポート番号を使用します。 HTTP が通常ポート 80 を使用するのに対し、HTTPS のデフォルトポート番号は 443 です。

LAN（ローカルエリアネットワーク）に接続されているクライアントからインターネットへの HTTP アクセスは、通常、プロキシと呼ばれる特別な HTTP サーバーを経由して処理されます。プロキシは LAN からインターネットへの出入口であり、セキュリティポリシーを実行し、キャッシュ、検証ルーチン、およびフィルタ機能を提供し、ファイアウォールとして機能します。HTTPS でセキュリティ保護されたインターネットアクセスのほとんどが、リモートサーバーとの接続を管理する、専用のプロキシサーバー経由で実行されます。このようなプロキシは「SSL プロキシ」 とも呼ばれています。

証明書は、証明書の所有者や発行者に関する情報、セッションキーデータの暗号化に使用する公開キー、有効期限、デジタル署名など、さまざまな情報項目が含まれたバイナリドキュメントです。 HTTPS サーバーによって提示される証明書は、通常、証明書チェーン全体で最も下のリンクです。このような証明書チェーンは、公開キーインフラストラクチャ（PKI）と呼ばれます。このようなチェーンの最上位にある証明書は、ルート証明書と呼ばれます。 ルート証明書は、一般に、認証機関（CA）と呼ばれる特別な組織によって発行されます。 CA によって発行および署名されたルート証明書も、CA（ルート）証明書と呼ばれます。 詳細については、*HTTP 開発者用のマニュアル*およびインターネットで公開されているその他の情報源を参照してください。

AT-TLS を使用した HTTPS

Natural `REQUEST DOCUMENT` ステートメントの HTTPS サポートは、z/OS の通信サーバーコンポーネント AT-TLS（Application Transparent-Transport Layer Security）がベースです。

AT-TLS は、TLS/SSL 暗号化をソケットアプリケーションの構成可能サービスとして提供します。 この機能は TCP/IP プロトコルスタックの上に追加層として実現され、SSL 機能をソケットアプリケーションに対するほとんどあるいは完全な透過（トランスペアレント）モードで利用します。 AT-TLS には 3 つの動作モードが用意されています。 『*z/OS Communications Server, IP Programmer's Guide and Reference. Version 1, Release 9*』（IBM マニュアル SC31-8787-09）の第 15 章を参照してください。

3 つのモードは次のとおりです。

- **Basic**

  ソケットアプリケーションは、変更されずに透過モードで実行され、AT-TLS 経由で暗号化された通信を実行していることが認識されません。このため、レガシーアプリケーションはソースコードを修正することなくセキュリティ保護されたモードで実行できます。

- **Aware**

  アプリケーションは、セキュリティ保護されたモードで実行されていることを認識し、TLS ステータス情報をクエリできます。

- **Controlling**

  ソケットアプリケーションは、AT-TLS を認識し、AT-TLS の暗号化サービスそのものの使用を制御します。 つまり、アプリケーションはセキュリティ保護された通信とそうでない通信を切り替えることができます。

Natural for Mainframes では *Controlling* モードを使用して、HTTPS 要求に対してのみセキュリティ保護されたモードをオンにし、HTTP 要求は暗号化しません。

z/OS での証明書の管理

z/OS では、AT-TLS で使用される証明書を 2 通りの方法で管理できます。 証明書は、z/OS UNIX ファイルシステム内の RACF キーリングまたはキーデータベースに格納されます。 どちらの方法が実際に適用されるかは、Natural HTTPS クライアントが使用する z/OS TCP/IP スタックの AT-TLS Policy Agent Configuration ファイルに定義されています。

IBM では、z/OS システムデリバリーごとに、一般的に使用される CA ルート証明書のセットを配布しています。 サーバー証明書の保持にキーリングを使用する予定の場合は、配布されたルート証明書をシステム管理者が手動でインポートする必要があります。 有効期限が切れたルート証明書の新しい置き換えが IBM から配布された場合は、影響を受けるすべてのキーリングを更新する必要があります。

キーデータベースはキーリングとは異なり、新たに作成されるとルート証明書の最新セットが自動的に格納されます。 ただし、キーデータベースを使用する場合も、ルート証明書の最新セットを常に維持することが必要です。

Natural HTTPS クライアントによって使用される証明書は、信頼済みとフラグされることが必要です。 証明書が公開キーインフラストラクチャの一部の場合は、対応する CA ルート証明書が信頼済みとフラグされることが必要です。

RACF キーリングの使用

RACF では、デジタル証明者はいわゆるキーリングに格納されます。 キーリングとそれが格納される証明書の作成と管理には、RACF コマンド `RACDCERT` が使用されます。

『*z/OS Security Server RACF Security Administrator's Guide*』（IBM マニュアル SA22-7683-11）および『*z/OS Security Server RACF Command Language Reference*』（IBM マニュアル SA22-7687-11）を参照してください。

キーデータベースの使用

RACF の代わりとして、z/OS UNIX サービスファイルシステムに存在するキーデータベースに証明書を格納できます。 キーデータベースの作成と管理には、GSKKYMAN ユーティリティを使用する必要があります。

『*z/OS Cryptographic Services PKI Services Guide and Reference*』（IBM マニュアル SA22-7693-10）を参照してください。

IMS/TM に関する制限事項

IMS/TM 環境で Natural ステートメント REQUEST DOCUMENT および PARSE XML を使用する場合は、以下の制限事項が適用されます。

- PARSE XML ステートメントは、TP モニタの IMS/TM で実行できますが、アクティブな PARSE ループ内で I/O ステートメントを実行できないという制限があります。 PARSE ループ内で I/O が発生すると、エラー NAT0967 が発行されます。

詳細については、ステートメントの説明の対応する注意点を参照してください。

openUTM での XML 関連ステートメントのサポートに対する前提条件

I/O の発生するアクティブな PARSE ループでは、UTM ファンクションコール PGWT を使用する必要があります。 これは以下のことを意味します。

1. UTMアプリケーションは、複数のタスクで開始する必要があります。そうでない場合、UTM エラー K319 が発生し、ダンプが生成されます。
2. PGWT 条件を KDCDEF に定義する必要があります。
   a. PGWT コール中の入力メッセージに対する最大待機時間（秒）を定義します。

      例：

      ```
      MAX PGWTTIME=60
      ```

   b. PGWT コールに対する最大の UTM タスク数を定義します。

例：

```
MAX TASKS-IN-PGWT=1
```

c. PGWTは、TAC-PRIORITIES命令またはTACCLASSコンセプトのどちらかを使用して制御できます。

- TAC-PRIORITIES命令を使用したPGWTの制御

  例：

```
DEFAULT TAC TYPE=D,PROGRAM=NATUTM,. . . . . .
TAC NAT,ADMIN=NO,TIME=(0,0),PGWT=YES,TACCLASS=1
TAC-PRIORITIES DIAL-PRIO=EQ
```

- TACCLASSコンセプトを使用したPGWTの制御

  例：

```
DEFAULT TAC TYPE=D,PROGRAM=NATUTM,. . . . . .
TAC NAT,ADMIN=NO,TIME=(0,0),TACCLASS=1
TAC NAT1,ADMIN=NO,TIME=(0,0),TACCLASS=2
TACCLASS 1,TASKS=2
TACCLASS 2,TASKS=1,PGWT=YES
```

3. パラメータマクロNURENTのサブパラメータILCSは、ILCS=CRTEに設定する必要があります。

サンプルプログラム

次のサンプルプログラムは、ステートメントREQUEST DOCUMENTおよびPARSE XMLの使用方法を示します。

これ以外のサンプルプログラムは、各ステートメントの説明の末尾とNaturalライブラリSYSEXVに用意されています。

```
DEFINE DATA
LOCAL
1 #FROM   (A) DYNAMIC
1 #HEADER (A) DYNAMIC
1 #PAGE   (A) DYNAMIC
1 #RC     (I4)
1 #COL    (N8)
1 #COL1   (I4)
1 #COL2   (I4)
1 #COL3   (I4)
```

```
1 #LOC    (A30)
1 #CP     (A) DYNAMIC
1 #PATH   (A) DYNAMIC
1 #NAME   (A) DYNAMIC
1 #VALUE  (A) DYNAMIC
1 #RTERR  (I4)
END-DEFINE
*
ASSIGN #FROM = 'HTTP://SI15.HQ.SAG/autos6.xml'
**
REQUEST DOCUMENT FROM #FROM
RETURN
HEADER ALL #HEADER
PAGE #PAGE    ENCODED    FOR TYPES 'TEXT/XML'
 CODEPAGE ' '
RESPONSE #RC
GIVING #RTERR
**
IF #RC NE 200 /* TEST FOR HTTP RESPONSE 200 = 'OK'
WRITE 'HTTP RESPONSE' #RC 'RECEIVED'
ESCAPE ROUTINE
END-IF
EJECT
PRINT #HEADER
/ '_'(79)
PRINT #PAGE
/ '_'(79)
/ '_'(79)
ASSIGN #CP = *CODEPAGE
EXAMINE #PAGE FOR 'encoding' GIVING POSITION  #COL1
 IF #COL1 GT 0
   EXAMINE #PAGE FOR '?>' GIVING POSITION #COL3
   IF #COL3 GT #COL1
     EXAMINE #PAGE FOR 'ISO-8859-1' GIVING POSITION #COL2
   END-IF
   IF #COL2 GT #COL1 AND #COL2 LT  #COL3
     EXAMINE #PAGE FOR 'ISO-8859-1' REPLACE #CP
   END-IF
 END-IF
PRINT #PAGE
/ '_'(79)
EJECT
PARSE XML #PAGE INTO PATH #PATH NAME #NAME VALUE #VALUE
 PRINT #PATH / 'NAME=' #NAME / 'VALUE=' #VALUE / '_'(79)
END-PARSE
END
```

注意: 上記のプログラムでアクセスされているURLはイントラネットサイトのアドレスを指しており、インターネットからはアクセスできません。

サンプルプログラムの出力：

```
HTTP/1.1 200 OK?Date: Thu, 10 Aug 2006 16:26:22 GMT?Server: Apache/1.3.19 (
BS2000)?Last-Modified: Thu, 27 Jul 2006 16:44:42 GMT?ETag: "2602c-111-44c8ed7a"
?Accept-Ranges: bytes?Content-Length: 273?Connection: close?Content-Type: text/
xml??
_____________________________________________________________________________
<?xml version="1.0" encoding="ISO-8859-1" ?><autos>?<make></make>?<make>Ford</
make>?<model>Thunderbird</model>?<make>Merceds-Benz</make><model>S400</model><
make>BWM</make><model version="latest">330I</model>?<make><label><company>
Mercedes</company></label></make>?</autos>?
_____________________________________________________________________________
<?xml version="1.0" encoding="IBM01140" ?><autos>?<make></make>?<make>Ford</
make>?<model>Thunderbird</model>?<make>Merceds-Benz</make><model>S400</model><
make>BWM</make><model version="latest">330I</model>?<make><label><company>
Mercedes</company></label></make>?</autos>?
_____________________________________________________________________________
```

MORE

```
autos
Name= autos
Value=
_____________________________________________________________________________
autos/$
Name=
Value= ?
_____________________________________________________________________________
autos/make
Name= make
Value=
_____________________________________________________________________________
autos/make//
Name= make
Value=
_____________________________________________________________________________
autos/$
Name=
Value= ?
_____________________________________________________________________________
autos/make
Name= make
Value=
VVVV
Name= autos
Value=
_____________________________________________________________________________
autos/$
Name=
Value= ?
_____________________________________________________________________________
autos/make
Name= make
```

```
Value=
_____________________________________________________________________________
```

よくある質問

コードページサポートを有効にする必要があるのはなぜですか。

メインフレーム上のNaturalに関するドキュメントに「「NaturalICUハンドラをNaturalニュークリアスにリンクする必要があります」」との記載があります。

PARSE XML ステートメント

コードページサポートが必要となります。これは、メインフレームプラットフォームでは、（すでにUTF-16でエンコードされている場合を除き、）解析されるドキュメントは常に内部的にUTF-16に変換されるためです。 ほとんどの場合、ドキュメントはUTF-16ではなく、変換が実行されます。 詳細については、PARSE XMLステートメントの説明および『Unicodeおよびコードページのサポート』ドキュメントのPARSE XMLの説明を参照してください。

REQUEST DOCUMENT ステートメント

受信HTTPヘッダーの解釈と送信HTTPヘッダーの変換には、ICUライブラリが必要です。 通常、受信ヘッダーはISO 8859-1でエンコードされており、メインフレームの場合は常にNaturalデフォルトコードページに変換することが必要です（システム変数*CODEPAGEも参照）。PCの場合、変換は必ずしも必要ではありません。

XML のキーワードサブパラメータ（RDP、RDNOP など）の使用方法を教えてください。

PC の場合、REQUEST DOCUMENT ステートメントは Internet Explorer を実行し、そこに定義されている設定を使用します。

メインフレームの場合は、すべての要求が経由してルーティングされる必要のある（イントラネット）プロキシサーバーの URL は、NTXML/XML のキーワードサブパラメータ RDP を使用して指定する必要があります。 キーワードサブパラメータ RDNOP を使用すると、プロキシサーバーを経由しない、直接アドレス指定するローカルドメイン（複数可）を定義できます。

サイトのプロキシサーバー、ポート番号、および HTTP サーバーの確認方法を教えてください。

サイトのプロキシサーバー、ポート番号、および HTTP サーバーについては、ネットワーク管理者に確認することが必要です。

サイトに定義されているプロキシサーバーは、ブラウザでも確認できます。

例えば、Internet Explorer では［ツール］>［インターネット オプション］>［接続］>［LAN の設定］>［詳細設定］で確認できます。

また、Web で検索するとこのような情報を確認できるツールが見つかることがあります。例（テストはしていません）：*http://www.sharewareconnection.com/titles/proxy-settings.htm*

TCP/IP または HTTP の問題と、Natural の問題とを区別する方法を教えてください。

HTTP レスポンスコード

HTTP レスポンスは、RESPONSE 節を介して *operand16* に返されます。 『ステートメント』ドキュメントの「*レスポンス番号の概要 - HTTP/HTTPS 要求*」に説明があります。

TCP/IP エラー

これらのエラーの範囲は 8300 番以降です。

特に、エラー NAT8304 では、失敗した HTTP 要求の詳細がわかります。

TCP/IP エラー番号はインストールされている環境によって異なることがあるため、NAT8304 によって返されるテキストが最も参考になります。

追加情報

- オフセット 480 のバッファ RDOCWA を参照してください。
- TCP/IP エラーはほとんどが ICU エラーです。プロファイルまたはセッションパラメータ CPCVERR を OFF に設定することをお勧めします。

メインフレームからWebサイトにNaturalなしでアクセスできるかどうかを確認する方法を教えてください。

問題がNaturalインストールに関連しているのか、より一般的な問題なのかを切り分けるには、TSOから PING を実行します。

例えば、TSOコマンドシェルに次のコマンドを入力します。

```
TSO PING www.google.com
```

次のような結果が返されます。

```
CS V1R9: Pinging host WWW.GOOGLE.COM (66.249.91.99)
Ping #1 response took 0.018 seconds.
```

次に、Naturalセッション内で、後述の小さいプログラムを使用してこのWebサイトへのアクセスをテストできます。

例えば、次のコマンドでNaturalを起動します。

```
NATvr CFICU=ON
XML=(ON,RDOC=ON,PARSE=ON,RDP='HTTPPROX.HQ.SAG',RDPPORT=8080,RDNOP='*.EUR.AD.SAG;
*.HQ.SAG;*.SOFTWAREAG.COM')
```

vr は、Naturalのリリースおよびバージョン番号を表します。

これらの値は、弊社の内部環境と保存されたプロファイルのものです。キーワードサブパラメータ RDP、RDPPORT、および RDNOP の設定をネットワーク管理者に確認するか、ブラウザ（Internet Explorer）に定義されている値を試すことが必要です。

次の処理を実行します。

```
DEFINE DATA LOCAL
1 #RESULTXML (A) DYNAMIC
1 #RC (I4)
END-DEFINE
REQUEST DOCUMENT FROM "HTTP://WWW.GOOGLE.DE"
RETURN HEADER ALL #HEADER RESPONSE #RC
WRITE #RC
WRITE #HEADER (AL=79)
END
```

NAT2TCP が正しくロードされているかどうか確認したいのですが。

SYSPROD ユーティリティで確認できます。

SYSPROD で、Natural 製品に対する SC（サブコンポーネントの表示）コマンドを入力します。 インストールされているサブコンポーネントのリストをスクロールしていくと、Nat Request Document のエントリ（Product ID .... TCP）が見つかります。

メッセージ「unsupported coding」が表示されます。

よくあるユーザーエラーです。XML ドキュメントは、暗黙的または明示的に、あるコードページから別のコードページに、例えば ISO-8859-1 からシステム変数 *CODEPAGE に指定されていたコードページに変換されます。 ただし、ドキュメントのエンコーディング PI encoding="ISO-8859-1" が、変換されたエンコーディングに合わせて調整されていません。 この場合、パーサは解析するドキュメントの最初の文字でエラーになり、終了します。

REQUEST DOCUMENT で Natural エラー NAT3411 が発生しないようにする方法を教えてください。

セッションパラメータ CPCVERR を OFF に設定します。

自己署名入りの証明書を使用できますか。

自己署名証明書は、OpenSSL の SDK を使用して、テストを目的としてイントラネットサーバーで使用できます。 キーデータベースまたは RACF キーリングにインポートした自己署名証明書は、信頼済みとフラグする必要があります。

証明書の管理に適している方法はどちらですか。

RACF キーリングのアプローチには、キーデータベースに比べてはるかに多くの作業が必要となります。 キーリングは HTTPS サーバーにアクセスするユーザーごとに作成する必要がある一方、キーデータベースは複数のユーザーで共有できます。

TCP/IP を AT-TLS 用に構成する方法を教えてください。

次のように進めます。

1. TCP/IP コンフィグレーションファイルで、TCPCONFIG ステートメントにオプション TTLS を設定します。
2. AT-TLS Policy Agent を構成および開始します。 このエージェントは、接続が SSL かどうかを確認するため、TCP/IP によって新しい TCP 接続のたびに呼び出されます。
3. AT-TLS ルールを含む Policy Agent ファイルを作成します。 Policy Agent ファイルには、どの接続が SSL かを規定するルールが含まれます。

『*z/OS Communications Server: IP Configuration Guide*』の第 18 章「*Application Transparent Transport Layer Security (AT-TLS) data protection*」も参照してください。

Sample Policy Agent ファイルには、すべての送信接続がアプリケーション制御の TLS として定義されています。 ルールがアプリケーション制御であるため、これにより Natural `REQUEST DOCUMENT` サポート以外の TCP/IP アプリケーションが影響されることはありません。 つまり、アプリケーションが接続ステータスを SSL に設定できます。 アプリケーションは、このステータスを設定しない限り、影響を受けません。一方、Policy Agent ファイルでは、アプリケーション制御の SSL 接続を特定のポート、ユーザー、またはアドレススペースに制限することもできます。 サンプルでは、証明書データベースが HFS ファイル `/ u/admin/CERT.kdb` に存在することが想定されています。

```
TTLSRule                          ConnRule01~1
{
  LocalAddrSetRef                 addr1
  RemoteAddrSetRef                addr1
  LocalPortRangeRef               portR1
  Direction                       Outbound
  Priority                        255
  TTLSGroupActionRef              gAct1~AllUsersAsClient
  TTLSEnvironmentActionRef        eAct1~AllUsersAsClient
  TTLSConnectionActionRef         cAct1~AllUsersAsClient
}
TTLSGroupAction                   gAct1~AllUsersAsClient
{
  TTLSEnabled                     On
  Trace                           6
}
TTLSEnvironmentAction             eAct1~AllUsersAsClient
{
  HandshakeRole                   Client
  EnvironmentUserInstance         0
  TTLSKeyringParmsRef             keyR1
}
TTLSConnectionAction              cAct1~AllUsersAsClient
{
  HandshakeRole                   Client
  TTLSCipherParmsRef              cipher1~AT-TLS__Silver
  TTLSConnectionAdvancedParmsRef  cAdv1~AllUsersAsClient
  Trace                           0
}
TTLSConnectionAdvancedParms       cAdv1~AllUsersAsClient
{
  ApplicationControlled           On
}
TTLSKeyringParms                  keyR1
{
  Keyring                         /u/admin/CERT.kdb
  KeyringStashFile                /u/admin/CERT.sth
}
```

```
TTLSCipherParms                     cipher1~AT-TLS__Silver
{
  V3CipherSuites                    TLS_RSA_WITH_DES_CBC_SHA
  V3CipherSuites                    TLS_RSA_WITH_3DES_EDE_CBC_SHA
  V3CipherSuites                    TLS_RSA_WITH_AES_128_CBC_SHA
}
IpAddrSet                           addr1
{
  Prefix                            0.0.0.0/0
}
PortRange                           portR1
{
  Port                              1024-65535
}
```

AT-TLS の構成を確認する方法を教えてください。

Policy-Agent ジョブ出力 JESMSGLG で次を確認します。

```
EZZ8771I PAGENT CONFIG POLICY PROCESSING COMPLETE FOR <your TCP/IP address space>:
TTLS
```

このメッセージは、初期化に成功したことを示しています。

Policy-Agent ジョブ出力 JESMSGLG で次を確認します。

```
EZZ8438I PAGENT POLICY DEFINITIONS CONTAIN ERRORS FOR <your TCP/IP address space>:
TTLS
```

このメッセージは、コンフィグレーションファイルにエラーがあることを示しています。 詳細については、syslog.log ファイルを参照してください。

コンフィグレーションルールはクライアントも対象となりますか。

syslog.log で次を確認します。

```
EZD1281I TTLS Map   CONNID: 00002909 LOCAL: 10.20.91.61..1751 REMOTE:
10.20.91.117..443
JOBNAME: KSP USERID: KSP TYPE: OutBound STATUS: Appl Control RULE: ConnRule01
ACTIONS: gAct1 eAct1 AllUsersAsClient
```

上記のエントリは、ポート 443 へのユーザー KSP による接続がアプリケーション制御であることを示しています。

問題の特定に関して他に情報はありませんか。

『*z/OS V1R8.0 Comm Svr: IP Diagnosis Guide: 3.23*』の第 29 章「*Diagnosing Application Transparent Transport Layer Security (AT-TLS)*」も参照してください。

ポリシーエージェントトレースに切り替える方法を教えてください。

『*Comm Svr: IP Configuration Reference*』の第 20 章「*Syslog deamon and Comm Svr*」および『*IP Configuration Guide*』の第 1.5.1 項「*Configuring the syslog daemon (syslogd)*」を参照してください。

接続確立時のエラー

ポリシーエージェントトレースで、リターンコード RC および対応する GSK_ ファンクション名を探します。

「*System SSL Programming*」の第 12.1 項「*SSL Function Return Codes*」の RC の説明を参照してください。

trace=255 の場合のサンプルトレース：

```
EZD1281I TTLS Map   CONNID: 00002909 LOCAL: 10.20.91.61..1751 REMOTE:
10.20.91.117..443 JOBNAME: KSP USERID: KSP TYPE: OutBound STATUS: A
EZD1283I TTLS Event GRPID: 00000003 ENVID: 00000000 CONNID: 00002909  RC:    0
Connection Init
EZD1282I TTLS Start GRPID: 00000003 ENVID: 00000002 CONNID: 00002909 Initial
Handshake ACTIONS: gAct1 eAct1 AllUsersAsClient HS-Client
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Call
GSK_SECURE_SOCKET_OPEN - 7EE4F718
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Set
GSK_SESSION_TYPE -  CLIENT
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Set
GSK_V3_CIPHER_SPECS -  090A2F
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Set
GSK_FD - 00002909
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Set
GSK_USER_DATA - 7EEE9B50
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:  435 Call
GSK_SECURE_SOCKET_INIT - 7EE4F718
EZD1283I TTLS Event GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:  435
Initial Handshake 00000000 7EEE8118
EZD1286I TTLS Error GRPID: 00000003 ENVID: 00000002 CONNID: 00002909 JOBNAME: KSP
USERID: KSP RULE: ConnRule01  RC:  435 Initial Handshake
EZD1283I TTLS Event GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0
Connection Close 00000000 7EEE8118
```

参照情報

役立つリソースのリストを以下に示します。

訓練コース

Software AGのコーポレートユニバーシティでは、このテーマの特別訓練コースを用意しています。 コーポレートユニバーシティの訓練コースについては、*http://servline24.softwareag.com/public/* の ServLine24 を参照してください。

また、お近くの地域におけるオンサイトの特別訓練コースについて、担当地域の Software AG にお問い合わせください。

有効なリンク

役立つリンクを以下に示します。

- World Wide Web Consortium（W3C）：*http://www.w3.org/*
- Extensible Markup Language（XML）：*http://www.w3.org/XML/*
- HyperText Markup Language（HTML）ホームページ：*http://www.w3.org/MarkUp/*
- W3 Schools：*http://www.w3schools.com/*

55 アプリケーションユーザーインターフェイスの設計

アプリケーションのユーザーインターフェイス、つまり、アプリケーションをどのようにユーザーに見せるかは、アプリケーションを作成する際の重要な検討項目です。

ここでは、Naturalが提供する、統一された外観を持ち、ユーザーへの指示やユーザーとの対話を行うための強力なメカニズムであるユーザーインターフェイスを設計するさまざまな方法について説明します。

ユーザーインターフェイスを設計するときは、標準および標準化が重要な要素となります。

Naturalを使用すると、さまざまなハードウェアおよびオペレーティングシステムで共通の機能をエンドユーザーに提供できます。

Natural には、全般的な画面レイアウト（情報、データ、メッセージの各エリア）、ファンクションキーの割り当て、およびウィンドウのレイアウトが含まれています。

このセクションでは、次のトピックについて説明します。

- 画面設計
- ダイアログ設計

56 画面設計

ファンクションキー行の制御 - 端末コマンド %Y

端末コマンド %Y を使用すると、Natural ファンクションキー行を表示する方法と位置を定義できます。

以下に参考情報を示します。

- ファンクションキー行のフォーマット
- ファンクションキー行の位置設定
- カーソル依存

ファンクションキー行のフォーマット

ファンクションキー行のフォーマットの定義には、以下の端末コマンドを使用できます。

%YN

Software AGのタブフォーマットでファンクションキー行を表示します。

```
 Command ===>

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       Help        Exit                                                  Canc
```

%YS

名前が割り当てられているキーのみをシーケンシャルに出力したファンクションキー行を表示します（PF1=*value*, PF2=*value*, ...）。

```
 Command ===>

 PF1=Help,PF3=Exit,PF12=Canc
```

%YP

名前が割り当てられているキーのみをPC風にシーケンシャルに出力したファンクションキー行を表示します（F1=*value*, F2=*value*, ...）。

```
Command ===>

F1=Help,F3=Exit,F12=Canc
```

その他の表示オプション

ファンクションキー行には、他にも以下のようなさまざまなコマンドオプションを使用できます。

- 1 行および 2 行表示
- 高輝度表示
- 反転表示
- カラー表示

これらのオプションの詳細については、『*端末コマンド*』ドキュメントの「*%Y - PF キー行の制御*」を参照してください。

ファンクションキー行の位置設定

%YB

画面の一番下にファンクションキー行を表示します。

```
16:50:53                    *****  NATURAL  *****                    2002-12-18
User SAG                        - Main Menu -                Library XYZ


                  Function

                _ Development Functions
                _ Development Environment Settings
                _ Maintenance and Transfer Utilities
                _ Debugging and Monitoring Utilities
                _ Example Libraries
                _ Other Products
                _ Help
                _ Exit Natural Session
```

```
Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                  Canc
```

%YT

画面の一番上にファンクションキー行を表示します。

```
 Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       Help        Exit                                                  Canc
 16:50:53                   *****  NATURAL  *****                  2002-12-18
 User SAG                        - Main Menu -                Library XYZ




                   Function



                 _ Development Functions

                 _ Development Environment Settings

                 _ Maintenance and Transfer Utilities

                 _ Debugging and Monitoring Utilities

                 _ Example Libraries

                 _ Other Products

                 _ Help

                 _ Exit Natural Session
```

```
Command ===>
```

%Y*nn*

画面の *nn* 行目にファンクションキー行を表示します。 以下の例では、ファンクションキー行は 10 行目に設定されています。

```
16:50:53                    *****  NATURAL  *****                    2002-12-18
User SAG                         - Main Menu -                  Library XYZ




                   Function



                 _ Development Functions

                 _ Development Environment Settings

                 _ Maintenance and Transfer Utilities

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---

       Help        Exit                                                   Canc
                  - Debugging and Monitoring Utilities
                 _ Example Libraries

                 _ Other Products

                 _ Help

                 _ Exit Natural Session
```

```
Command ===>
```

カーソル依存

%YC

このコマンドにより、ファンクションキー行がカーソル依存になります。これは、ファンクションキー行が PC 画面上のアクションバーに似た動作をすることを意味します。希望するファンクションキー番号にカーソルを移動するか、またはファンクションキー名を指定して Enter キーを押すと、対応するファンクションキーが押されたかのような動作が Natural によって行われます。

カーソル依存をオフにするには、`%YC` を再度入力します（トグル切り替え）。

ファンクションキー名のみを表示して（`%YH`）タブ表示フォーマット（`%YN`）とともに `%YC` を使用することにより、快適なアクションバーの処理をアプリケーションで実現できます。単にファンクション名をカーソルで選択して Enter キーを押すだけで、そのファンクションを実行できます。

メッセージ行の制御 - 端末コマンド %M

端末コマンド `%M` には、Natural メッセージ行を表示する方法と位置を定義するためのさまざまなオプションを使用できます。

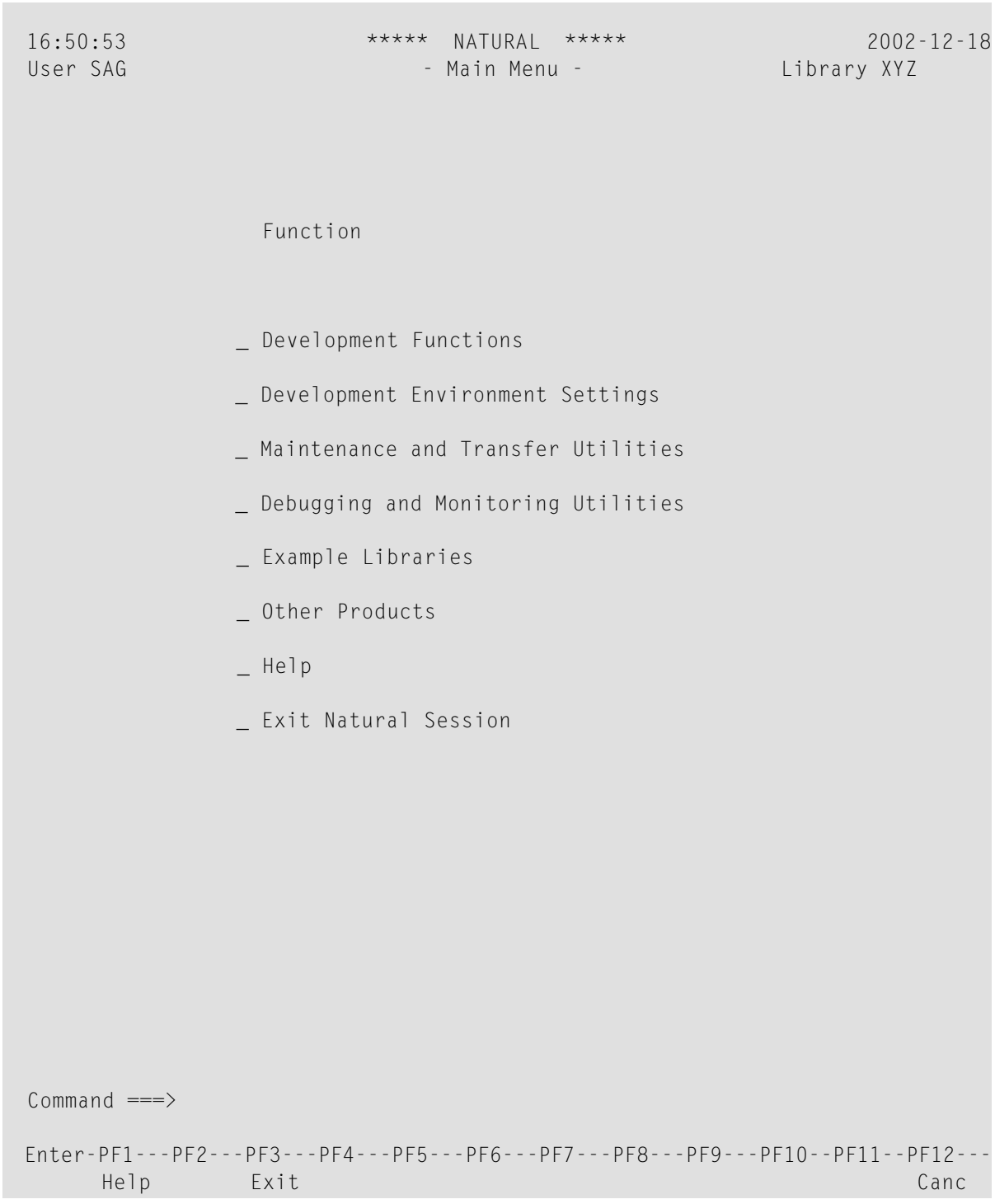
以下に参考情報を示します。

- メッセージ行の位置設定
- メッセージ行の保護
- メッセージ行の色

メッセージ行の位置設定

%MB

画面の一番下にメッセージ行を表示します。



```
Please enter a function.
```

%MT

画面の一番上にメッセージ行を表示します。

```
Please enter a function.

16:50:53                    *****  NATURAL  *****                   2002-12-18
User SAG                         - Main Menu -                 Library XYZ




                    Function



                  _ Development Functions

                  _ Development Environment Settings

                  _ Maintenance and Transfer Utilities

                  _ Debugging and Monitoring Utilities

                  _ Example Libraries

                  _ Other Products

                  _ Help

                  _ Exit Natural Session









Command ===>
```

```
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
       Help         Exit
```

メッセージ行の位置設定に関する他のオプションについては、『*端末コマンド*』ドキュメントの「*%M - メッセージ行の制御*」を参照してください。

メッセージ行の保護

%MP
メッセージ行を非保護モードから保護モードに、またはその逆に切り替えます。非保護モードでは、メッセージ行も端末入力に使用できます。

メッセージ行の色

%M=*color-code*
指定した色でメッセージ行を表示します。カラーコードの詳細については、『*パラメータリファレンス*』ドキュメントに記載されている、セッションパラメータ CD の説明を参照してください。

フィールドへの色の割り当て - 端末コマンド %=

端末コマンド %= を使用すると、もともと色の設定をサポートしていないプログラムのフィールド属性に色を割り当てることができます。 コマンドにより、指定の属性で定義されたすべてのフィールド／テキストが、指定の色で表示されます。

定義済みの色割り当てが使用している端末タイプに合わない場合は、このコマンドを使用して元の割り当てを新しい割り当てで上書きできます。

%= コマンドは、Naturalエディタ内で、例えばマップの作成中に色割り当てをダイナミックに定義するために使用することもできます。

コード	説明
空白	色変換テーブルをクリアします。
F	新たに定義した色で、プログラムで割り当てた色を上書きします。
N	プログラムで割り当てられた色属性は変更されません。
O	出力フィールド
M	変更可能なフィールド（出力および入力）
T	テキスト定数
B	点滅
C	斜体
D	デフォルト値

コード	説明
I	高輝度
U	下線付き
V	反転
BG	背景
BL	青
GR	緑
NE	デフォルト色
PI	ピンク
RE	赤
TU	空色
YE	黄色

例：

```
%=TI=RE,OB=YE
```

この例では、すべての高輝度テキストフィールドに赤を割り当て、すべての点滅出力フィールドに黄色を割り当てています。

取り囲み - 端末コマンド %D=B

取り囲み（外枠の縁取り）とは、端末画面に特定のフィールドが表示されたときにそのフィールドの周囲に線を生成する機能です。 フィールドの周囲にそのような「外枠」を描画することは、フィールドの長さと画面上での位置をユーザーに示すためのもう 1 つのメソッドです。

取り囲み機能は、通常、ダブルバイト文字セットの表示もサポートしている特定のタイプの端末でしか使用できません。

端末コマンド `%D=B` は、取り囲みの制御に使用されます。 このコマンドの詳細については、『*端末コマンド*』ドキュメントの関連するセクションを参照してください。

統計行／情報行 - 端末コマンド %X

端末コマンド %X は、Natural 統計行／情報行の表示を制御します。 この行は統計行または情報行として使用できますが、同時に両方を兼ねることはできません。

以下に参考情報を示します。

- 統計行
- 情報行

統計行

統計行の表示／非表示を切り替えるには、端末コマンド %X を入力します（これはトグル機能です）。 統計行を表示に設定すると、以下のような統計情報が表示されます。

- 前回の画面操作で画面に転送されたバイト数
- 現在のページの論理行サイズ
- 現在のページの物理行サイズ

統計行の詳細については、『*端末コマンド*』ドキュメントに記載されている、端末コマンド %X の説明を参照してください。

以下の例は、画面の一番下に表示されている統計行を示しています。

```
>                                        > +  Program     POS      Lib SAG
All    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 SET CONTROL 'XT'
  0020 SET CONTROL 'XI+'
  0030 DEFINE PRINTER (2) OUTPUT 'INFOLINE'
  0040 WRITE (2) 'EXECUTING' *PROGRAM 'BY' *INIT-USER
  0050 WRITE 'TEST OUTPUT'
  0070 END
  0080
  0090
  0100
  0110
  0120
  0130
  0140
  0150
  0160
  0170
  0180
  0190
```

```
    0200
IO=264,AI =292,L=0 C= ,LS=80,P =23,PLS=80,PCS=24,FLD=82,CLS=1,ADA=0
```

情報行

統計行は、例えば、デバッグのためにステータス情報を表示する*情報行*として使用することも、SAA 標準による定義に従って、セパレータ行として使用することもできます。

統計行を情報行として定義するには、端末コマンド %XI+ を使用します。

上記のコマンドで情報行を有効にすると、以下の例のように、DEFINE PRINTER ステートメントを使用したデータの出力先として情報行を定義できます。

```
SET CONTROL 'XT'
SET CONTROL 'XI+'
DEFINE PRINTER (2) OUTPUT 'INFOLINE'
WRITE (2) 'EXECUTING' *PROGRAM 'BY' *INIT-USER
WRITE 'TEST OUTPUT'
END
```

上記のプログラムを実行すると、出力画面の一番上の情報行にステータス情報が表示されます。

```
EXECUTING POS      BY SAG
Page      1                                              2001-01-22  10:56:06

TEST OUTPUT
```

統計行／情報行の詳細については、『*端末コマンド*』ドキュメントに記載されている、端末コマンド %X の説明を参照してください。

ウィンドウ

以下に参考情報を示します。

- ウィンドウとは
- DEFINE WINDOW ステートメント

- INPUT WINDOW ステートメント

ウィンドウとは

*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。

*論理*ページとは、Natural の出力エリアのことです。つまり、論理ページには、Natural プログラムによって表示用に生成された現在のレポート／マップが含まれています。 この論理ページは物理画面よりも大きくすることができます。

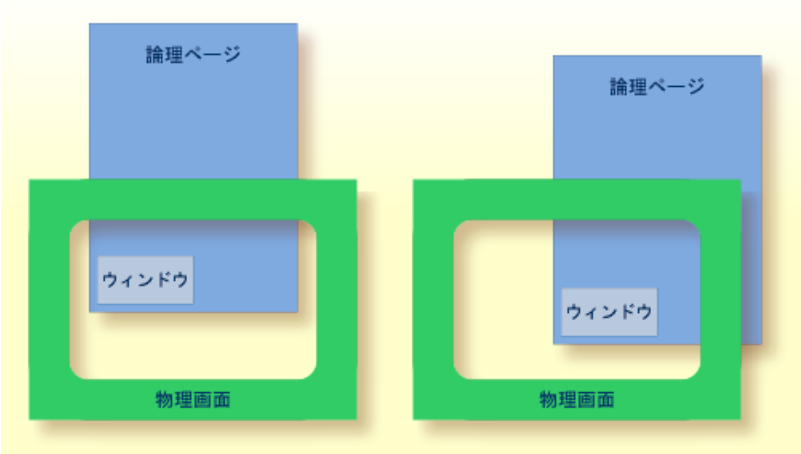
存在に気付かない場合もありますが、ウィンドウは常に存在しています。 DEFINE WINDOW ステートメントで個別に指定されない限り、ウィンドウのサイズは端末画面の物理サイズと同一です。

ウィンドウは、以下の 2 つの方法で操作できます。

- *物理画面*上では、ウィンドウのサイズと位置を制御できます。
- *論理*ページ上では、ウィンドウの位置を制御できます。

物理画面上の位置

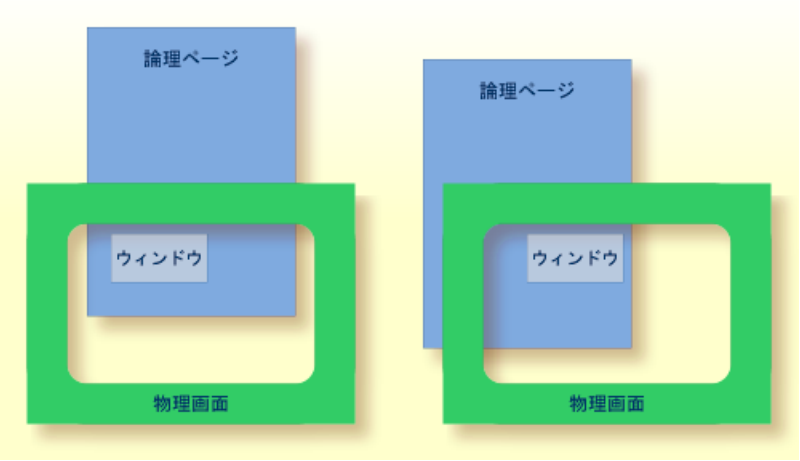
以下の図は、物理画面上のウィンドウの位置を示しています。 どちらの例にも、画面上のウィンドウの位置が違うだけで、論理ページの同じセクションが表示されていることに注意してください。



論理ページ上の位置

以下の図は、論理ページ上のウィンドウの位置を示しています。

*論理*ページでウィンドウの位置を変更しても、*物理画面*上のウィンドウのサイズおよび位置は変更されません。 つまり、ウィンドウがページ上で移動するのではなく、ページがウィンドウの「下」 で移動するということです。



DEFINE WINDOW ステートメント

DEFINE WINDOW ステートメントを使用して、*物理画面*上のウィンドウのサイズ、位置、属性を指定します。

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。

DEFINE WINDOW ステートメントには、さまざまなオプションを使用できます。 これらのオプションについて、以下の例で説明します。 このステートメントは、Naturalのデフォルトの端末タイプ設定を参照します。端末コマンド %T= およびプロファイルパラメータ TTYPE の説明も参照してください。

以下のプログラムは、物理画面上のウィンドウの位置を定義しています。

```
** Example 'WINDX01': DEFINE WINDOW
************************************************************************
DEFINE DATA LOCAL
1 COMMAND (A10)
END-DEFINE
*
DEFINE WINDOW TEST
       SIZE 5*25
       BASE 5/40
       TITLE 'Sample Window'
       CONTROL WINDOW
       FRAMED POSITION SYMBOL BOTTOM LEFT
```

```
*
INPUT WINDOW='TEST' WITH TEXT 'message line'
      COMMAND (AD=I'_') /
      'dataline 1' /
      'dataline 2' /
      'dataline 3' 'long data line'
*
IF COMMAND = 'TEST2'
  FETCH 'WINDX02'
ELSE
  IF COMMAND = '.'
    STOP
  ELSE
    REINPUT 'invalid command'
  END-IF
END-IF
END
```

ウィンドウは、ウィンドウ名によって識別されます。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。この例では、ウィンドウ名は TEST です。

ウィンドウのサイズは、SIZE オプションで設定します。この例では、ウィンドウの高さは 5 行、幅は 25 列（ポジション）です。

ウィンドウの位置は、BASE オプションで設定します。この例では、ウィンドウの左上隅は行 5、列 40 に位置します。

TITLE オプションを使用すると、ウィンドウフレームに表示するタイトルを定義できますが、これはウィンドウのフレームを定義した場合にのみ有効です。

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。この例では、CONTROL WINDOW によって、ウィンドウ内にメッセージ行が表示されます。CONTROL SCREEN を使用すると、ウィンドウの外側の物理画面全体に行が表示されます。CONTROL 節を省略すると、デフォルトで CONTROL WINDOW が適用されます。

FRAMED オプションを使用すると、フレーム付きのウィンドウを定義できます。このフレームは、カーソル依存です。カーソル依存が適用可能な場所では、適切な記号（<、-、+、>。POSITION 節の説明を参照）の上にカーソルを置いて Enter キーを押すだけで、ウィンドウ内のページを上下左右に移動できます。つまり、物理画面上のウィンドウの下にある*論理ページ*を移動できます。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。

FRAMED オプションの POSITION 節を使用して、ウィンドウのフレームに表示する、論理ページ上のウィンドウの位置情報を定義します。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 を使用すると、位置情報が記号形式（「More: <-+>」）で表示されます。 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 で、位置情報を上のフレームラインまたは下のフレームラインのどちらに表示するかを指定します。

LEFT／RIGHT で、位置情報をフレームラインの左または右のどちらに表示するかを指定します。

端末コマンド %F=*chv* で、フレームに使用する文字を定義できます。

c	最初の文字は、ウィンドウフレームの 4 つの*隅*に使用されます。
h	2 番目の文字は、*水平*フレームラインに使用されます。
v	3 番目の文字は、*垂直*フレームラインに使用されます。

例：

```
%F=+-!
```

上記のコマンドを実行すると、以下のようなウィンドウフレームが生成されます。

```
+------------------------+
!                        !
!                        !
!                        !
!                        !
+------------------------+
```

INPUT WINDOW ステートメント

INPUT WINDOW ステートメントは、DEFINE WINDOW ステートメントで定義したウィンドウを有効にします。 以下の例では、ウィンドウ TEST が有効化されています。 ウィンドウにデータを出力する場合（WRITE ステートメントを使用する場合など）は、SET WINDOW ステートメントを使用します。

前述のプログラムを実行すると、入力フィールド COMMAND がウィンドウに表示されます。 セッションパラメータ AD を使用して、フィールドの値を高輝度で表示し、下線を充填文字として使用するよう定義されています。

プログラム WINDX01 の出力：

```
> r                                    > +  Program     WINDX01  Lib SYSEXPG
Top    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 ** Example 'WINDX01': DEFINE WINDOW
  0020 ****************************** +----Sample Window-----+ ***************
  0030 DEFINE DATA LOCAL               ! message line         !
  0040 1 COMMAND (A10)                 ! COMMAND ___________  !
  0050 END-DEFINE                      ! dataline 1           !
  0060 *                               +More:     + >---------+
  0070 DEFINE WINDOW TEST
  0080        SIZE 5*25
  0090        BASE 5/40
  0100        TITLE 'Sample Window'
  0110        CONTROL WINDOW
  0120        FRAMED POSITION SYMBOL BOTTOM LEFT
  0130 *
  0140 INPUT WINDOW='TEST' WITH TEXT 'message line'
  0150       COMMAND (AD=I'_') /
  0160       'dataline 1' /
  0170       'dataline 2' /
  0180       'dataline 3' 'long data line'
  0190 *
  0200 IF COMMAND = 'TEST2'
       ....+....1....+....2....+....3....+....4....+....5....+... S 29   L 1
```

下のフレームラインの位置情報 `More: + >` は、ウィンドウに表示されている情報の他にも論理ページ上に情報があることを示しています。

論理ページ上のさらに下にある情報を表示するには、カーソルを下のフレームラインのプラス記号（+）に移動して Enter キーを押します。

ウィンドウの表示が下に移動します。テキスト `long data line` はウィンドウのサイズに収まらないため、完全には表示されていないことに注意してください。

```
> r                                    > +  Program     WINDX01  Lib SYSEXPG
Top    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 ** Example 'WINDX01': DEFINE WINDOW
  0020 ****************************** +----Sample Window-----+ ***************
  0030 DEFINE DATA LOCAL               ! message line         !
  0040 1 COMMAND (A10)                 ! dataline 3 long data !
  0050 END-DEFINE                      ! dataline 2           !
  0060 *                               +More:   -   >---------+
  0070 DEFINE WINDOW TEST
  0080        SIZE 5*25
  0090        BASE 5/40
  0100        TITLE 'Sample Window'
  0110        CONTROL WINDOW
  0120        FRAMED POSITION SYMBOL BOTTOM LEFT
  0130 *
  0140 INPUT WINDOW='TEST' WITH TEXT 'message line'
  0150       COMMAND (AD=I'_') /
```

```
  0160       'dataline 1' /
  0170       'dataline 2' /
  0180       'dataline 3' 'long data line'
  0190 *
  0200 IF COMMAND = 'TEST2'
       ....+....1....+....2....+....3....+....4....+....5....+... S 29   L 1
```

この表示されていない情報を表示するには、カーソルを下のフレームラインの記号>に移動して Enter キーを押します。 ウィンドウの表示が論理ページ上の右に移動して、それまで表示されていなかった単語 `line` が表示されます。

```
> r                                    > +  Program     WINDX01  Lib SYSEXPG
Top    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 ** Example 'WINDX01': DEFINE WINDOW
  0020 ****************************** +----Sample Window-----+ ***************
  0030 DEFINE DATA LOCAL               ! message line         !
  0040 1 COMMAND (A10)                 !  line                ! <==
  0050 END-DEFINE                      !                      !
  0060 *                               +More: < -    ---------+
  0070 DEFINE WINDOW TEST
  0080        SIZE 5*25
  0090        BASE 5/40
  0100        TITLE 'Sample Window'
  0110        CONTROL WINDOW
  0120        FRAMED POSITION SYMBOL BOTTOM LEFT
  0130 *
  0140 INPUT WINDOW='TEST' WITH TEXT 'message line'
  0150       COMMAND (AD=I'_') /
  0160       'dataline 1' /
  0170       'dataline 2' /
  0180       'dataline 3' 'long data line'
  0190 *
  0200 IF COMMAND = 'TEST2'
       ....+....1....+....2....+....3....+....4....+....5....+... S 29   L 1
```

複数のウィンドウ

複数のウィンドウを開くことはもちろん可能です。ただし、Naturalウィンドウでは、アクティブにできるのは常に1つのウィンドウのみ、つまり最新のウィンドウのみです。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。

COMMAND フィールドに「TEST2」と入力すると、プログラム WINDX02 が実行されます。

```
** Example 'WINDX02': DEFINE WINDOW
************************************************************************
DEFINE DATA LOCAL
1 COMMAND (A10)
END-DEFINE
*
DEFINE WINDOW TEST2
       SIZE 5*30
       BASE 15/40
       TITLE 'Another Window'
       CONTROL SCREEN
       FRAMED POSITION SYMBOL BOTTOM LEFT
*
INPUT WINDOW='TEST2' WITH TEXT 'message line'
      COMMAND (AD=I'_') /
      'dataline 1' /
      'dataline 2' /
      'dataline 3' 'long data line'
*
IF COMMAND = 'TEST'
  FETCH 'WINDX01'
ELSE
  IF COMMAND = '.'
    STOP
  ELSE
    REINPUT 'invalid command'
  END-IF
END-IF
END
```

2つ目のウィンドウが開きます。もう1つのウィンドウは表示されたままですが、非アクティブです。

```
message line
> r                                      > +  Program     WINDX01  Lib SYSEXPG
Top    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 ** Example 'WINDX01': DEFINE WINDOW
  0020 ****************************** +----Sample Window-----+ ***************
  0030 DEFINE DATA LOCAL               ! message line         !  Inactive
  0040 1 COMMAND (A10)                 ! COMMAND TEST2_____   !  Window
  0050 END-DEFINE                      ! dataline 1           !  <==
  0060 *                               +More:     + >---------+
  0070 DEFINE WINDOW TEST
  0080        SIZE 5*25
  0090        BASE 5/40
  0100        TITLE 'Sample Window'
```

```
0110        CONTROL WINDOW
0120       FRAMED POSITION SYMBOL B +------Another Window-------+  Currently

0130 *                               ! COMMAND ___________         !  Active
0140 INPUT WINDOW='TEST' WITH TEXT ' ! dataline 1                  !  Window
0150       COMMAND (AD=I'_') /       ! dataline 2                  !  <==
0160       'dataline 1' /            +More:   +------------------+
0170       'dataline 2' /
0180       'dataline 3' 'long data line'
0190 *
0200 IF COMMAND = 'TEST2'
```

新しいウィンドウのメッセージ行は、ウィンドウ上ではなく、物理画面の上部に表示されることに注意してください。 これは、WINDX02 プログラムの CONTROL SCREEN 節によって定義されています。

DEFINE WINDOW、INPUT WINDOW、SET WINDOW の各ステートメントの詳細については、『*ステートメント*』ドキュメントの対応する説明を参照してください。

標準／ダイナミックレイアウトマップ

標準レイアウトマップ

「*マップエディタのチュートリアル*」セクションの説明のとおり、*標準レイアウト*は、マップエディタで定義できます。 このレイアウトによって、アプリケーション全体のすべてのマップの外観を統一できます。

標準レイアウトを参照するマップを初期化すると、その標準レイアウトはマップに固定的に組み込まれます。 つまり、標準レイアウトを変更する場合、変更を有効にするために影響を受けるマップをすべて再カタログ化する必要があります。

ダイナミックレイアウトマップ

標準レイアウトとは対照的に、*ダイナミックレイアウト*は、そのレイアウトを参照するマップに固定的に組み込まれるのではなく、ランタイム時に組み込まれます。

したがって、レイアウトマップをマップエディタの［マップエディタプロファイル（Define Map Settings For MAP）］画面で「ダイナミック」として定義すると（以下の例を参照）、レイアウトに対する変更はすべて、そのレイアウトを参照するすべてのマップに引き継がれます。マップを再カタログ化する必要はありません。

```
08:46:18               Define Map Settings for MAP                  2001-01-22

 Delimiters           Format                          Context
 -----------------    ----------------------------    -------------------------
 Cls Att CD  Del      Page Size ..... 23              Device Check .... ________
  T   D      BLANK    Line Size ..... 79              WRITE Statement   _
  T   I      ?        Column Shift ... 0 (0/1)        INPUT Statement   X
  A   D      _        Layout ......... STAN1___       Help  ____________________
  A   I      )         dynamic ....... Y (Y/N)         as field default N (Y/N)
  A   N      ª        Zero Print ..... N (Y/N)
  M   D      &        Case Default ... UC (UC/LC)
  M   I      :        Manual Skip .... N (Y/N)        Automatic Rule Rank  1
  O   D      +        Decimal Char ... .              Profile Name .... SYSPROF
  O   I      (        Standard Keys .. Y (Y/N)
                      Justification .. L (L/R)        Filler Characters
                      Print Mode ..... __             ------------------------
                                                      Optional, Partial .... _
                      Control Var .... ________       Required, Partial .... _
                                                      Optional, Complete ... _
 Apply changes only to new fields?     N (Y/N)        Required, Complete ... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                   Let
```

レイアウトマップの詳細については、『エディタ』ドキュメントの「マップエディタ」を参照してください。

多言語ユーザーインターフェイス

Natural を使用すると、各国で使用するための多言語アプリケーションを作成できます。

マップ、ヘルプルーチン、エラーメッセージ、プログラム、サブプログラム、コピーコードは、ダブルバイト文字セットを使用する言語を含め、最大 60 の異なる言語で定義できます。

以下に参考情報を示します。

- 言語コード
- Natural オブジェクトの言語の定義
- ユーザー言語の定義
- 多言語オブジェクトの参照
- プログラム

- エラーメッセージ
- 日付／時刻フィールドの編集マスク

言語コード

Naturalでは、言語ごとに*言語コード*（1～60）が割り当てられています。以下の表は、言語コード表を一部抜粋したものです。 言語コードの完全な表については、『システム変数』ドキュメントに記載されている、システム変数 *LANGUAGE の説明を参照してください。

言語コード	言語	オブジェクト名のマップコード
1	英語	1
2	ドイツ語	2
3	フランス語	3
4	スペイン語	4
5	イタリア語	5
6	オランダ語	6
7	トルコ語	7
8	デンマーク語	8
9	ノルウェー語	9
10	アルバニア語	A
11	ポルトガル語	B

言語コード（左の列）が、システム変数 *LANGUAGE に含まれているコードです。 このコードはNaturalによって内部的に使用されます。 このコードを使用してユーザー言語を定義します（以下の「*ユーザー言語の定義*」を参照）。 表の右の列のマップコードを使用して、Natural オブジェクトの言語を識別します。

例：

ポルトガルの言語コードは 「11」 です。 ポルトガル語版の Natural オブジェクトをカタログ化するときに使用するコードは 「B」 です。

言語コードの完全な表については、『システム変数』ドキュメントに記載されている、システム変数 *LANGUAGE の説明を参照してください。

Natural オブジェクトの言語の定義

Natural オブジェクト（マップ、ヘルプルーチン、プログラム、サブプログラム、コピーコード）の言語を定義するには、対応するマップコードをオブジェクト名に追加します。マップコードを除くオブジェクト名は、すべての言語で同一である必要があります。

以下の例では、マップが英語とドイツ語で作成されています。マップの言語を識別するために、各言語に対応するマップコードがマップ名に組み込まれています。

多言語アプリケーションのマップ名の例

`DEMO1` = 英語のマップ（マップコード 1）

`DEMO2` = ドイツ語のマップ（マップコード 2）

英字のマップコードを持つ言語の定義

マップコードは、1〜9、A〜Z または a〜y の範囲内の値です。 英字のマップコードには、特別な操作が必要です。

小文字は大文字に自動的に変換されるため、通常は、名前に小文字があるオブジェクトをカタログ化することはできません。

しかし、例えば、言語コードが 59 でマップコードが `x` の漢字（日本語）をオブジェクト名に定義する場合、小文字を使用する必要があります。

このようなオブジェクトをカタログ化するには、まず正しい言語コード（ここでは 59）を端末コマンド `%L=nn` を使用して設定します。*nn* は言語コードです。

その後、実際のマップコードの代わりにアンパーサンド（&）文字をオブジェクト名に使用して、オブジェクトをカタログ化します。 したがって、マップ `DEMO` の日本語版を作成するには、`DEMO&` という名前でマップを STOW します。

ここで Natural オブジェクトのリストを確認すると、マップは `DEMOx` として正しくリストされます。

言語コード 1〜9 および大文字の A〜Z のオブジェクトは、アンパーサンド（&）表記を使用せずに直接カタログ化できます。

以下の例では、3 つのマップ DEMO1、DEMO2、および DEMOx を確認できます。 マップ DEMOx を削除するには、作成時と同じ方法で、つまり、端末コマンド %L=59 で正しい言語を設定し、アンパーサンド（&）表記（DEMO&）を使用して削除します。

```
08:41:14               ***** NATURAL LIST COMMAND *****              2001-01-25
User SAG             LIST * *                                    Library SAG

Cmd  Name      Type          S/C  SM  Vers  Level  User-ID   Date      Time
---  --------  ------------  ---  --  ----  -----  --------  --------  --------
__   COM3      Program       S/C  S   2.2   0002   SAG       92-01-21  14:34:39
__   CUR       Program  +--------- DELETE ---------+         92-01-22  09:37:02
__   CURS      Map      !                           !         92-01-22  09:37:41
__   D         Program  !  Please confirm deletion !         92-01-21  14:13:14
__   DARL      Program  !  with name DEMOx          !         91-06-03  12:08:30
__   DARL1     Local    !             DEMO&___      !         91-06-03  12:03:52
__   DAV       Program  +---------------------------+         92-01-29  09:07:52
de   DEMOx     Map           S/C  S   2.2   0002   SAG       92-02-25  08:41:04
__   DEMO1     Map           S/C  S   2.2   0002   SAG       92-01-22  08:38:32
__   DEMO2     Map           S/C  S   2.2   0002   SAG       92-01-22  08:07:32
__   DOWNCOM   Program       S    S   2.2   0001   SAG       91-08-12  14:01:10
__   DOWNCOMR  Program       S    S   2.2   0001   SAG       91-08-12  14:01:32
__   DOWNCOM2  Program       S    S   2.2   0001   SAG       91-08-15  13:02:20
__   DOWNDIR   Program       S    S   2.2   0001   SAG       91-08-16  08:03:56
From ________ (New start value)                                    0

Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit              --    -     +                      Canc
```

ユーザー言語の定義

システム変数 *LANGUAGE の定義として、プロファイルパラメータ ULANG（『パラメータリファレンス』を参照）または端末コマンド %L=*nn*（*nn* は言語コード）を使用して、ユーザーごとに使用する言語を定義します。

多言語オブジェクトの参照

多言語オブジェクトをプログラムで参照するには、オブジェクト名にアンパーサンド（&）文字を使用します。

以下のプログラムでは、マップ DEMO1 および DEMO2 が使用されています。 マップ名の最後のアンパーサンド（&）文字はマップコードを表し、*LANGUAGE システム変数で定義されている現在の言語のマップを使用することを意味します。

```
DEFINE DATA LOCAL
1 PERSONNEL VIEW OF EMPLOYEES
  2 NAME (A20)
  2 PERSONNEL-ID (A8)
1 CAR VIEW OF VEHICLES
   2 REG-NUM (A15)
1 #CODE (N1)
END-DEFINE
*
INPUT USING MAP 'DEMO&' /* <--- INVOKE MAP WITH CURRENT LANGUAGE CODE
...
```

このプログラムを実行すると、英語のマップ（DEMO1）が表示されます。 これは、*LANGUAGE の現在の値が 1 = 英語であるためです。

```
                                                    MAP DEMO1

                              SAMPLE MAP



                    Please select a function!



                    1.) Employee information

                    2.) Vehicle information



                    Enter code here: _
```

以下の例では、端末コマンド %L=2 を使用して、言語コードが 2= ドイツ語に変更されています。

プログラムを再度実行すると、ドイツ語のマップ（DEMO2）が表示されます。

```
                              BEISPIEL-MAP



                    Bitte wählen Sie eine Funktion!


                    1.) Mitarbeiterdaten

                    2.) Fahrzeugdaten


                 Code hier eingeben: _
```

プログラム

アプリケーションの中には、多言語プログラムを定義すると便利なものがあります。 例えば、一般的な請求書発行プログラムでは、請求書を発行する国ごとのさまざまな税制度に対応する、複数のサブプログラムを使用します。

多言語プログラムは、前述のマップの場合と同様の方法で定義します。

エラーメッセージ

Natural ユーティリティ SYSERR を使用すると、Natural エラーメッセージを最大 60 の言語に変換できます。また、独自のエラーメッセージを定義することもできます。

ユーザーに表示されるメッセージの言語は、*LANGUAGE システム変数によって決まります。

エラーメッセージの詳細については、『*ユーティリティ*』ドキュメントの「*SYSERR ユーティリティ*」を参照してください。

日付／時刻フィールドの編集マスク

編集マスクを使用して定義する日付フィールドおよび時刻フィールドに使用される言語もまた、システム変数 *LANGUAGE によって決まります。

編集マスクの詳細については、『*パラメータリファレンス*』に記載されている、セッションパラメータ EM の説明を参照してください。

スキル別ユーザーインターフェイス

同じアプリケーションを使用しつつ、ユーザーのスキルレベルに応じて（さまざまな詳細度の）異なるマップを使用したい場合があります。

アプリケーションを多言語で国際的に使用する必要がない場合、多言語のマップに対する手法を使用して、詳細度の異なるマップを定義できます。

例えば、初心者用に言語コード 1 を定義し、上級者用に言語コード 2 を定義できます。 この単純かつ有効な手法を以下に示します。

以下のマップ（PERS1）には、エンドユーザー向けに、メニュー機能を選択する方法の説明が含まれています。 非常に詳細な情報が記載されています。 マップ名には、マップコード 1 が含まれています。

```
                                                    MAP PERS1

                              SAMPLE MAP

                    Please select a function


                    1.) Employee information  _

                    2.) Vehicle information   _

                        Enter code:  _

  To select a function, do one of the following:

  - place the cursor on the input field next to desired function and press Enter
  - mark the input field next to desired function with an X and press Enter
  - enter the desired function code (1 or 2) in the 'Enter code' field and press
Enter
```

詳細な説明が含まれていない同じマップが、同じ名前で保存されています。ただし、マップコードは 2 です。

```
                                                MAP PERS2


                          SAMPLE MAP

                Please select a function


                1.) Employee information  _

                2.) Vehicle information   _

                    Enter code:  _
```

上記の例では、ULANG プロファイルパラメータの値が 1 の場合は詳細な説明付きのマップが出力され、値が 2 の場合は説明なしのマップが出力されます。『*パラメータリファレンス*』に記載されている、プロファイルパラメータ ULANG の説明も参照してください。

57 ダイアログ設計

この章では、ユーザーとアプリケーションとの対話を簡単かつ柔軟に行うユーザーインターフェイスの設計方法について説明します。

フィールドに基づいた処理

*CURS-FIELD および POS(field-name)

システム変数 *CURS-FIELD をシステム関数 POS(*field-name*) とともに使用すると、Enter キーが押された時点でカーソルが位置付けられているフィールドに基づいた処理を定義できます。

*CURS-FIELD には、カーソルが現在位置付けられている入力フィールドの内部 ID が格納されます。この ID は単体では使用できません。POS(*field-name*) と組み合わせてのみ使用できます。

*CURS-FIELD および POS(*field-name*) は、例えば、特定のフィールドにカーソルを配置して Enter キーを押すだけで機能を選択できるようにするために使用できます。

以下の例は、そのようなアプリケーションを示しています。

```
DEFINE DATA LOCAL
1 #EMP (A1)
1 #CAR (A1)
1 #CODE (N1)
END-DEFINE
*
INPUT USING MAP 'CURS'
*
DECIDE FOR FIRST CONDITION
  WHEN *CURS-FIELD = POS(#EMP) OR #EMP = 'X' OR #CODE = 1
    FETCH 'LISTEMP'
  WHEN *CURS-FIELD = POS(#CAR) OR #CAR = 'X' OR #CODE = 2
    FETCH 'LISTCAR'
   WHEN NONE
    REINPUT 'PLEASE MAKE A VALID SELECTION'
END-DECIDE
END
```

結果は以下のとおりです。

```
                                SAMPLE MAP

                    Please select a function


                    1.) Employee information  _
                    2.) Vehicle information   _  <== Cursor positioned
                                                      on field

                        Enter code:  _

  To select a function, do one of the following:

  - place the cursor on the input field next to desired function and press Enter
  - mark the input field next to desired function with an X and press Enter
  - enter the desired function code (1 or 2) in the 'Enter code' field and press
Enter
```

ユーザーがカーソルを［Employee information］の隣の入力フィールド（#EMP）に配置して Enter キーを押すと、プログラム LISTEMP によって従業員名のリストが表示されます。

```
  Page      1                                                2001-01-22  09:39:32

          NAME
  --------------------

  ABELLAN
  ACHIESON
  ADAM
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  AECKERLE
  AFANASSIEV
  AFANASSIEV
  AHL
  AKROYD
```

注意: *CURS-FIELD および POS(*field-name*) の値は、フィールドの内部 ID の取得にのみ使用します。 算術演算には使用できません。

プログラミングの単純化

システム関数 POS

Natural システム関数 POS(*field-name*) は、指定された名前のフィールドの内部 ID を返します。

POS(*field-name*) を使用すると、マップ内の位置と関係なく、特定のフィールドを特定できます。 これは、マップ内のフィールドのシーケンスと数が変更されても、POS(*field-name*) は引き続き同じフィールドを一意に識別できることを意味します。 これにより、例えば、プログラムロジックに依存しているとフィールドに MARK を付ける場合、必要なのは 1 つの REINPUT ステートメントのみになります。

注意: POS(*field-name*) の値は、フィールドの内部 ID の取得にのみ使用します。 算術演算には使用できません。

例：

```
...
DECIDE ON FIRST VALUE OF ...
   VALUE ...
      COMPUTE #FIELDX = POS(FIELD1)
   VALUE ...
      COMPUTE #FIELDX = POS(FIELD2)
   ...
END-DECIDE
...
REINPUT ... MARK #FIELDX
...
```

*CURS-FIELD および POS(*field-name*) の詳細については、『システム変数』ドキュメントおよび『システム*関数*』ドキュメントを参照してください。

行に基づいた処理

システム変数 *CURS-LINE

システム変数 *CURS-LINE を使用すると、Enter キーが押された時点でカーソルが位置付けられている行に基づいた処理を作成できます。

この変数を使用することにより、ユーザーが使いやすいメニューを作成できます。 適切なプログラミングが行われていれば、ユーザーは実行したいメニューオプションの行にカーソルを移動して Enter キーを押すだけで、そのオプションを実行できます。

カーソルの位置は、画面上の物理的な位置に関係なく、その時点のアクティブなウィンドウ内の位置で定義されます。

注意: メッセージ行、ファンクションキー行、および情報行／統計行は、画面上のデータ行とは見なされません。

以下の例は、*CURS-LINE システム変数を使用した、行に基づいた処理を示しています。マップ上で Enter キーが押されると、オプション [Employee information] がある 8 行目にカーソルが配置されているかどうかがプログラムによって確認されます。8行目にカーソルがある場合、従業員名をリストするプログラム LISTEMP が実行されます。

```
DEFINE DATA LOCAL
1 #EMP (A1)
1 #CAR (A1)
1 #CODE (N1)
END-DEFINE
*
INPUT USING MAP 'CURS'
*
DECIDE FOR FIRST CONDITION
  WHEN *CURS-LINE = 8
     FETCH 'LISTEMP'
   WHEN NONE
     REINPUT 'PLACE CURSOR ON LINE OF OPTION YOU WISH TO SELECT'
END-DECIDE
END
```

出力：

```
                                Company Information


                    Please select a function


                 [] 1.) Employee information

                    2.) Vehicle information









    Place the cursor on the line of the option you wish to select and press
    Enter
```

[] で表示されているカーソルを希望するオプションに移動して Enter キーを押すと、対応するプログラムが実行されます。

列に基づいた処理

システム変数 *CURS-COL

システム変数 *CURS-COL は、前述の *CURS-LINE と同様に使用できます。 *CURS-COL を使用すると、画面上でカーソルが位置付けられている列に基づいた処理を作成できます。

ファンクションキーに基づいた処理

システム変数 *PF-KEY

ユーザーが押したファンクションキーに応じて処理を行う必要がある場合は少なくありません。

これは、ステートメント SET KEY とシステム変数 *PF-KEY を使用し、デフォルトのマップ設定（Standard Keys = Y）を変更することによって実現できます。

SET KEY ステートメントは、プログラムの実行中に機能をファンクションキーに割り当てます。システム変数 *PF-KEY には、ユーザーが最後に押したファンクションキーの ID が格納されます。

以下の例は、*PF-KEY と組み合わせた SET KEY の使い方を示しています。

```
...
SET KEY PF1
*
INPUT USING MAP 'DEMO&'
IF *PF-KEY = 'PF1'
  WRITE 'Help is currently not active'
END-IF
...
```

SET KEY ステートメントによって、PF1 キーがファンクションキーとして有効化されています。

IF ステートメントは、ユーザーが PF1 キーを押したときに実行するアクションを定義しています。 システム変数 *PF-KEY の現在の値を確認し、その値が PF1 の場合は、対応するアクションを実行します。

ステートメント SET KEY およびシステム変数 *PF-KEY の詳細については、それぞれ『ステートメント』ドキュメントおよび『システム変数』ドキュメントを参照してください。

ファンクションキー名に基づいた処理

システム変数 *PF-NAME

ファンクションキーに基づいた処理を定義する場合、システム変数 *PF-NAME を使用するとさらに快適に定義できます。 この変数を使用すると、特定のキーではなく、ファンクションの名前に応じた処理を作成できます。

変数 *PF-NAME には、ユーザーが最後に押したファンクションキーの名前、つまり、SET KEY ステートメントの NAMED 節でキーに割り当てた名前が格納されます。

例えば、PF3 キーまたは PF12 キーが押されたらヘルプが起動されるようにする場合、両方のキーに同じ名前（以下の例では INFO）を割り当てます。 ユーザーがこれらのどちらかのキーを押すと、IF ステートメントで定義した処理が実行されます。

```
...
SET KEY PF3  NAMED 'INFO'
        PF12 NAMED 'INFO'
INPUT USING MAP 'DEMO&'
IF *PF-NAME = 'INFO'
  WRITE 'Help is currently not active'
END-IF
...
```

NAMED 節で定義したファンクション名はファンクションキー行に表示されます。

```
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
                  INFO                                                    INFO
```

アクティブなウィンドウの外部からのデータ処理

以下に参考情報を示します。

- システム変数 *COM
- *COM の使用例

- *COM へのカーソルの配置 - %T* 端末コマンド

システム変数 *COM

「*画面設計*」の「*ウィンドウ*」で説明されているとおり、アクティブになれるウィンドウは常に *1* つだけです。これは、通常は、その特定のウィンドウ内でのみ入力が可能だということです。

通信エリアと見なされる *COM システム変数を使用すると、アクティブなウィンドウの外でデータを入力できます。

ただし、マップに *COM が変更可能なフィールドとして含まれていることが前提条件です。この条件が満たされていれば、ウィンドウが画面上に表示されているときは、このフィールドにデータを入力できます。 *COM の内容に応じて、追加の処理を実行できます。

これにより、入力フィールドを持つウィンドウがアクティブな場合でも常にユーザーがコマンド行にデータを入力できる、Software AG のオフィスシステム Con-nect ですでに使用されているようなユーザーインターフェイスを実装できます。

*COM は、Natural セッションが終了するときにのみクリアされます。

*COM の使用例

以下の例では、プログラム ADD は、マップの入力データを使用して単純な加算処理を実行しています。このマップでは、拡張フィールド編集の AL フィールドで指定されている長さの変更可能なフィールドとして、（マップの最後に）*COM は定義されています。計算結果はウィンドウに表示されます。このウィンドウには入力できませんが、ウィンドウ外部のマップ内では引き続き *COM フィールドを使用できます。

プログラム **ADD**：

```
DEFINE DATA LOCAL
1 #VALUE1 (N4)
1 #VALUE2 (N4)
1 #SUM3 (N8)
END-DEFINE
*
DEFINE WINDOW EMP
  SIZE 8*17
  BASE 10/2
  TITLE 'Total of Add'
  CONTROL SCREEN
  FRAMED POSITION SYMBOL BOT LEFT
*
INPUT USING MAP 'WINDOW'
*
COMPUTE #SUM3 = #VALUE1 + #VALUE2
*
```

```
SET WINDOW 'EMP'
INPUT (AD=O) / 'Value 1 +' /
               'Value 2 =' //
               ' ' #SUM3
*
IF *COM = 'M'
  FETCH 'MULTIPLY' #VALUE1 #VALUE2
END-IF
END
```

プログラム ADD の出力：

```
                  Map to Demonstrate Windows with *COM

                                 CALCULATOR

                       Enter values you wish to calculate


                       Value 1:  12__
                       Value 2:  12__
   +-Total of Add-+
   !              !
   ! Value 1 +    !
   ! Value 2 =    !
   !              !
   !          24  !
   !              !
   +--------------+


    Next line is input field (*COM) for input outside the window:
```

この例では、値Mを入力すると、乗算が開始されます。入力マップの2つの値を乗算した結果が2つ目のウィンドウに表示されます。

```
                  Map to Demonstrate Windows with *COM

                                 CALCULATOR

                       Enter values you wish to calculate


                       Value 1:  12__
                       Value 2:  12__
   +-Total of Add-+                              +--------------+
   !              !                              !              !
   ! Value 1 +    !                              ! Value 1 x    !
```

```
   ! Value 2 =    !                                ! Value 2 =    !
   !              !                                !              !
   !          24  !                                !       144    !
   !              !                                !              !
   +--------------+                                +--------------+


    Next line is input field (*COM) for input outside the window:
                 M
```

COM へのカーソルの配置 - %T 端末コマンド

通常、アクティブなウィンドウが入力フィールド（AD=A、または AD=M）を持っていない場合、カーソルはウィンドウの左上隅に配置されます。

アクティブなウィンドウに入力フィールドがない場合、端末コマンド %T* を使用すると、ウィンドウの外にあるシステム変数 *COM にカーソルを配置できます。

%T* を再度使用すると、標準のカーソル位置に戻せます。

例：

```
...
INPUT USING MAP 'WINDOW'
*
COMPUTE #SUM3 = #VALUE1 + #VALUE2
*
SET CONTROL 'T*'
SET WINDOW 'EMP'
INPUT (AD=O) / 'Value 1 +' /
               'Value 2 =' //
               ' ' #SUM3
...
```

画面からのデータのコピー

以下に参考情報を示します。

- 端末コマンド %CS および %CC

端末コマンド %CS および %CC

これらの端末コマンドを使用すると、画面の一部を Natural スタック（%CS）またはシステム変数 *COM（%CC）にコピーできます。 特定の画面行の保護データがフィールドごとにコピーされます。

これらの端末コマンドのすべてのオプションについては、『*端末コマンド*』ドキュメントを参照してください。

スタックまたは *COM にコピーされたデータは、その後の処理で使用できます。 これらのコマンドを使用すると、以下の例のような、使いやすいインターフェイスを作成できます。

後の処理で使用する、レポート出力の行の選択

以下の例では、プログラム COM1 は、Abellan から Alestia までのすべての従業員名をリストしています。

プログラム **COM1**：

```
DEFINE DATA LOCAL
1 EMP VIEW OF EMPLOYEES
  2 NAME(A20)
  2 MIDDLE-NAME (A20)
  2 PERSONNEL-ID (A8)
END-DEFINE
*
READ EMP BY NAME STARTING FROM 'ABELLAN' THRU 'ALESTIA'
  DISPLAY NAME
END-READ
FETCH 'COM2'
END
```

プログラム COM1 の出力：

```
  Page      1                                          2006-08-12  09:41:21

           NAME
  --------------------

  ABELLAN
  ACHIESON
  ADAM
  ADKINSON
  ADKINSON
```

```
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  AECKERLE
  AFANASSIEV
  AFANASSIEV
  AHL
  AKROYD
  ALEMAN
  ALESTIA
  MORE
```

プログラムの制御は COM2 に移されています。

プログラム **COM2**：

```
DEFINE DATA LOCAL
1 EMP VIEW OF EMPLOYEES
  2 NAME(A20)
  2 MIDDLE-NAME (A20)
  2 PERSONNEL-ID (A8)
1 SELECTNAME (A20)
END-DEFINE
*
SET KEY PF5 = '%CCC'
*
INPUT NO ERASE 'SELECT FIELD WITH CURSOR AND PRESS PF5'
*
MOVE *COM TO SELECTNAME
FIND EMP WITH NAME = SELECTNAME
  DISPLAY NAME PERSONNEL-ID
END-FIND
END
```

このプログラムでは、端末コマンド %CCC が PF5 に割り当てられています。 この端末コマンドは、カーソルが配置されている行のすべての保護データをシステム変数 *COM にコピーします。 コピーされた情報は、その後の処理で使用できます。 この処理は、太字で記述されているプログラム行に定義されています。

任意の名前にカーソルを合わせて PF5 キーを押すと、詳しい従業員情報が出力されます。

```
SELECT FIELD WITH CURSOR AND PRESS PF5                      2006-08-12  09:44:25

          NAME
  --------------------

  ABELLAN
  ACHIESON
  ADAM  <==  Cursor positioned on name for which more information is required
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  AECKERLE
  AFANASSIEV
  AFANASSIEV
  AHL
  AKROYD
  ALEMAN
  ALESTIA
```

この場合、選択した従業員の個人 ID が表示されます。

```
  Page      1                                                 2006-08-12  09:44:52

          NAME          PERSONNEL
                           ID
  -------------------- ---------

  ADAM                 50005800
```

REINPUT／REINPUT FULL ステートメント

INPUT ステートメントに戻って再度実行する必要がある場合、REINPUT ステートメントを使用します。 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。

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。
- カーソルを特定のフィールドに配置する場合、MARK オプションを使用できます。また、特定のフィールド内の特定の位置にカーソルを配置する場合は、MARK POSITION オプションを使用します。

以下の例は、MARK POSITION を使用した REINPUT FULL の使い方を示しています。

```
DEFINE DATA LOCAL
1 #A (A10)
1 #B (N4)
1 #C (N4)
END-DEFINE
*
INPUT (AD=M) #A #B #C
IF #A = ' '
  COMPUTE #B = #B + #C
  RESET #C
  REINPUT FULL 'Enter a value' MARK POSITION 5 IN *#A
END-IF
END
```

フィールド #B に「3」、フィールド #C に「3」と入力して、Enter キーを押します。

```
  #A            #B      3 #C      3
```

プログラムは、フィールド #A に空白以外の値を必要とします。 MARK POSITION 5 IN *#A を使用した REINPUT FULL ステートメントによって、制御が入力画面に戻ります。COMPUTE で計算が実行されているため、変更可能なフィールド #B に値 6 が設定されています。 カーソルはフィールド #A の 5 番目のポジションに配置され、すぐに入力できるようになっています。

```
Enter name of field
  #A    _       #B      6 #C      0



Enter a value
```

以下の画面は、FULL オプションを使用しなかった場合に、同じステートメントによって返される画面です。 変数 #B および #C が INPUT ステートメントの実行時の状態に戻されている（各フィールドの値が 3）ことに注意してください。

```
#A     _          #B       3 #C       3
```

オブジェクト指向の処理 - Natural コマンドプロセッサ

Naturalコマンドプロセッサは、アプリケーション内のナビゲーション（移動操作）を定義および制御するために使用します。 Natural コマンドプロセッサは、開発部分とランタイム部分の 2 つの部分によって構成されています。

- 開発部分は、ユーティリティ SYSNCP です。 このユーティリティを使用すると、コマンド、およびこれらのコマンドの実行に対応するアクションを定義できます。 SYSNCP は、管理者が設定した定義により、コマンド入力時に実行する処理を判断するデシジョンテーブルを生成します。
- ランタイム部分は、ステートメント PROCESS COMMAND です。 このステートメントは、Natural プログラムからコマンドプロセッサを呼び出すために使用します。 このステートメントで、データ入力時の操作に使用する SYSNCP テーブルの名前を指定します。

Natural コマンドプロセッサの詳細については、『ユーティリティ』ドキュメントの「*SYSNCP ユーティリティ*」、および『ステートメント』ドキュメントの「PROCESS COMMAND」を参照してください。

58 NaturalX

ここでは、オブジェクトベースのアプリケーションの開発方法について説明します。

次のトピックについて説明します。

- **NaturalX** について
- **NaturalX** アプリケーションの開発

59 NaturalX について

この章では、NaturalX インターフェイスおよび専用の Natural ステートメントセットを使用するコンポーネントベースのプログラミングについて簡単に紹介します。

なぜ NaturalX か

コンポーネントアーキテクチャに基づくソフトウェアアプリケーションには、従来の設計よりも多くの利点があります。 例えば、次のような利点があります。

- 迅速な開発。 プログラマは、事前に構築されたコンポーネントからソフトウェアをアセンブルすることによって、アプリケーションをより速く構築できます。
- 開発費の削減。 プログラムのインターフェイスの共通のセットを用意することによって、コンポーネントを完全なソリューションに統合するための労力が少なくなります。
- 柔軟性の向上。 アプリケーションを構成しているコンポーネントの一部を交換するだけで、社内の各部門に合わせたソフトウェアのカスタマイズが容易になります。
- メンテナンスコストの削減。 アップグレードの際は、多くの場合、アプリケーション全体を修正するのではなく、コンポーネントの一部を交換するだけで済みます。

NaturalX を使用すると、コンポーネントベースのアプリケーションを作成できます。

NaturalX を使用することによって、コンポーネントベースのプログラミングスタイルを適用できます。 ただし、このプラットフォームではコンポーネントを配布できないため、実行できるのはローカルの Natural セッション内のみになります。

60 NaturalX アプリケーションの開発

この章では、クラスを定義および使用してアプリケーションを開発する方法について説明します。

開発環境

- **Windows** プラットフォームでのクラスの開発
  Windows プラットフォームでは、Natural クラスを開発するためのツールとして、Natural からクラスビルダが提供されます。 クラスビルダは、構造化された階層順に Natural クラスを表示し、ユーザーがクラスとそのコンポーネントを効率的に管理できるようにします。 クラスビルダを使用すると、この後で説明する構文要素に関する知識はまったく必要ないか、または基礎知識だけで済みます。
- **SPoD** を使用したクラスの作成
  メインフレーム、UNIX、または OpenVMS の一部またはすべてのリモート開発サーバーが含まれる Natural Single Point of Development（SPoD）環境では、Natural スタジオのフロントエンドで使用可能なクラスビルダを使用して、メインフレーム、UNIX、または OpenVMS プラットフォーム上でクラスを開発できます。 この場合、この後で説明する構文要素に関する知識はまったく必要ないか、または基礎知識だけで済みます。
- メインフレーム、**UNIX**、または **OpenVMS** プラットフォームでのクラスの開発
  SPoD を使用しない場合は、Natural プログラムエディタを使ってこれらのプラットフォームのクラスを開発することができます。 この場合は、この後で説明するクラス定義の構文の知識が必要です。

クラスの定義

クラスを定義するときは、Natural クラスモジュールを作成する必要があります。この中で `DEFINE CLASS` ステートメントを作成します。 `DEFINE CLASS` ステートメントを使用して、外部的に使用可能な名前をクラスに割り当て、そのインターフェイス、メソッドおよびプロパティを定義します。 クラスのインスタンスのレイアウトを記述するオブジェクトデータエリアをクラスに割り当てることもできます。

このセクションでは、次のトピックについて説明します。

- メソッドの実装

Natural クラスモジュールの作成

手順 60.1. Natural クラスモジュールを作成するには

- CREATE OBJECT ステートメントを使用して、クラスタイプの Natural オブジェクトを作成します。

クラスの指定

DEFINE CLASS ステートメントは、クラスの名前、クラスがサポートするインターフェイス、およびそのオブジェクトの構造を定義します。

手順 60.2. クラスを指定するには

- DEFINE CLASS ステートメントを使用します。詳細については『ステートメント』ドキュメントを参照してください。

インターフェイスの定義

クラスの各インターフェイスは、クラス定義内部の INTERFACE ステートメントで指定されます。INTERFACE ステートメントは、インターフェイスの名前と多数のプロパティおよびメソッドを指定します。COM クラスとして登録されるクラスについては、インターフェイスのグローバルユニーク ID も指定されます。

クラスには 1 つ以上のインターフェイスを定義できます。 インターフェイスごとに 1 つの INTERFACE ステートメントがクラス定義に指定されます。 各 INTERFACE ステートメントには 1 つ以上の PROPERTY 節および METHOD 節が含まれます。 通常、1 つのインターフェイスに含まれるプロパティとメソッドは、技術またはビジネスのいずれかの観点から関連付けられます。

PROPERTY 節はプロパティの名前を定義し、オブジェクトデータエリアからプロパティに変数を割り当てます。 この変数を使用して、プロパティの値を保存します。

METHOD 節はメソッドの名前を定義し、メソッドにサブプログラムを割り当てます。 このサブプログラムを使用して、メソッドを実装します。

手順 60.3. インターフェイスを定義するには

- INTERFACE ステートメントを使用します。詳細については『ステートメント』ドキュメントを参照してください。

オブジェクトデータ変数のプロパティへの割り当て

PROPERTY ステートメントは、複数のクラスが同じインターフェイスを異なる方法で実装するときにのみ使用されます。 この場合、クラスは同じインターフェイス定義を共有し、Natural コピーコードからインクルードします。次に PROPERTY ステートメントを使用して、インターフェイス定義の外でオブジェクトデータエリアからプロパティに変数を割り当てます。 INTERFACE ステートメントの PROPERTY 節と同様に、PROPERTY ステートメントはプロパティの名前を定義し、オブジェクトデータエリアからプロパティに変数を割り当てます。 この変数を使用して、プロパティの値を保存します。

手順 60.4. オブジェクトデータ変数をプロパティに割り当てるには

- PROPERTY ステートメントを使用します。詳細については『ステートメント』ドキュメントを参照してください。

サブプログラムのメソッドへの割り当て

METHOD ステートメントは、複数のクラスが同じインターフェイスを異なる方法で実装するときにのみ使用されます。 この場合、クラスは同じインターフェイス定義を共有し、Natural コピーコードからインクルードします。 METHOD ステートメントは、インターフェイス定義の外でサブプログラムをメソッドに割り当てるために使用されます。 INTERFACE ステートメントの METHOD 節と同様に、METHOD ステートメントはメソッドの名前を定義し、サブプログラムをメソッドに割り当てます。 このサブプログラムを使用して、メソッドを実装します。

手順 60.5. サブプログラムをメソッドに割り当てるには

- METHOD ステートメントを使用します。詳細については『ステートメント』ドキュメントを参照してください。

メソッドの実装

メソッドは次のような一般形式の Natural サブプログラムとして実装されます。

```
DEFINE DATA statement
*
* Implementation code of the method
*
END
```

DEFINE DATA ステートメントの詳細については、『ステートメント』ドキュメントを参照してください。

DEFINE DATA ステートメントのすべての節はオプションです。

データの整合性を確保するために、インラインデータ定義ではなくデータエリアを使うことをお勧めします。

PARAMETER 節を指定すると、メソッドにパラメータや戻り値を指定することができます。

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 とマークされていないパラメータは、「BY REFERENCE」によって渡される入力／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。

OPTIONAL としてマークされているパラメータは、Natural バージョン 4.1 以降のすべてのリリースで使用可能です。 オプションのパラメータは、メソッドが呼び出される場合は指定する必要はありません。 SEND METHOD ステートメントで *n*X 表記を使用することにより、これらを未指定にしておくことができます。

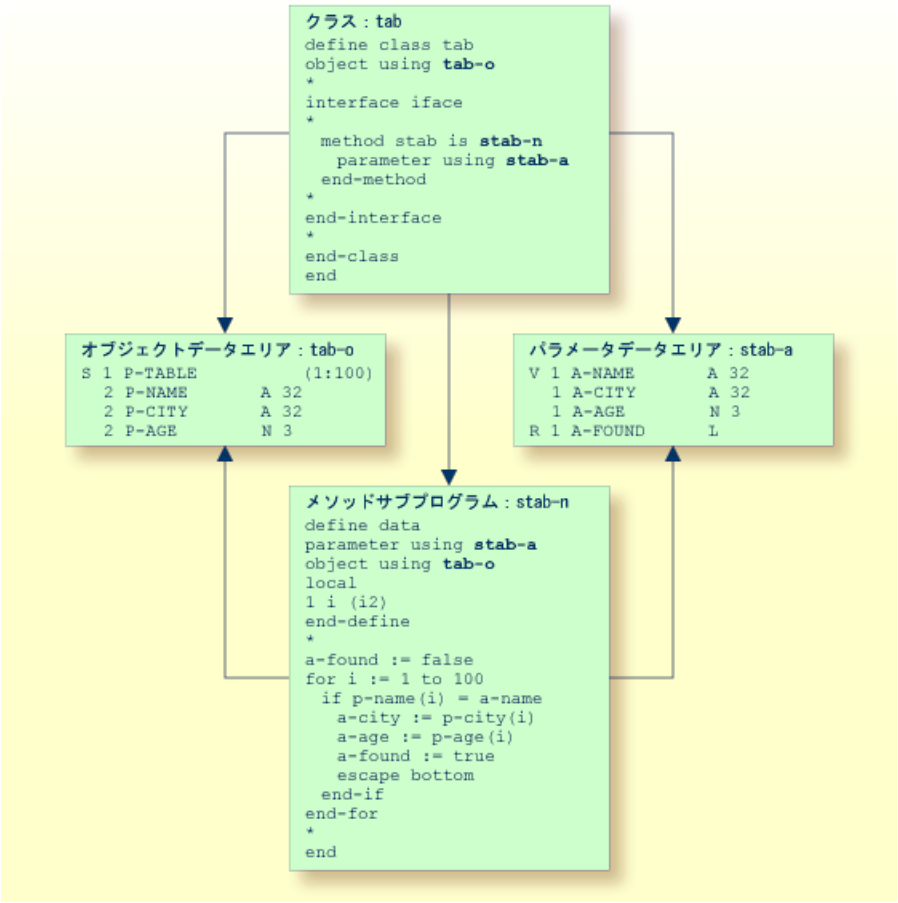
メソッドサブプログラムが、クラス定義内の対応する METHOD ステートメントに指定されたものとまったく同じパラメータを受け入れるようにするには、インラインデータ定義ではなくパラメータデータエリアを使用します。 対応する METHOD ステートメント内のものと同じパラメータデータエリアを使用します。

メソッドサブプログラムでオブジェクトデータ構造にアクセスするために、OBJECT 節を指定できます。 メソッドサブプログラムがオブジェクトデータに正しくアクセスできるようにするには、インラインデータ定義ではなくローカルデータエリアを使用します。 DEFINE CLASS ステートメントの OBJECT 節に指定されているものと同じローカルデータエリアを使用してください。

GLOBAL 節、LOCAL 節、および INDEPENDENT 節は、他の任意の Natural プログラムで使用できます。

技術的には可能ですが、通常はメソッドサブプログラムで CONTEXT 節を使用することは意味がありません。

次の例は、特定の人物に関するデータをテーブルから検索します。 検索キーは BY VALUE パラメータとして渡されます。 結果のデータは、「BY REFERENCE」 パラメータで返されます。 「BY REFERENCE」はデフォルト定義です。 メソッドの戻り値は BY VALUE RESULT 指定によって定義されます。



クラスとオブジェクトの使用

ローカルの Natural セッションで作成されたオブジェクトは、同じ Natural セッションの他のモジュールからアクセスできます。

CREATE OBJECT ステートメントは、特定のクラスのオブジェクト（インスタンスとも呼ばれます）を作成するために使用します。

Natural プログラムのオブジェクトを参照するには、オブジェクトハンドルを DEFINE DATA ステートメントで定義する必要があります。 オブジェクトのメソッドは SEND METHOD ステートメントで呼び出されます。 オブジェクトにはプロパティを指定できます。プロパティには通常の割り当て構文を使用してアクセスできます。

次の手順について説明します。

- オブジェクトハンドルの定義
- クラスのインスタンスの作成
- オブジェクトの特定メソッドの呼び出し

- プロパティへのアクセス

オブジェクトハンドルの定義

Natural プログラムのオブジェクトを参照するには、オブジェクトハンドルを DEFINE DATA ステートメントで次のように定義する必要があります。

```
DEFINE DATA
  level-handle-name [(array-definition)] HANDLE OF OBJECT
  ...
END-DEFINE
```

例：

```
DEFINE DATA LOCAL
1 #MYOBJ1 HANDLE OF OBJECT
1 #MYOBJ2 (1:5) HANDLE OF OBJECT
END-DEFINE
```

クラスのインスタンスの作成

手順 60.6. クラスのインスタンスを作成するには

- CREATE OBJECT ステートメントを使用します。詳細については『ステートメント』ドキュメントを参照してください。

オブジェクトの特定メソッドの呼び出し

手順 60.7. オブジェクトの特定メソッドを呼び出すには

- SEND METHOD ステートメントを使用します。詳細については『ステートメント』ドキュメントを参照してください。

プロパティへのアクセス

プロパティには、次のように ASSIGN（または COMPUTE）ステートメントを使用してアクセスできます。

```
ASSIGN operand1.property-name = operand2
ASSIGN operand2 = operand1.property-name
```

オブジェクトハンドル - *operand1*

operand1 はオブジェクトハンドルとして定義する必要があり、プロパティがアクセスされるオブジェクトを識別します。 オブジェクトがすでに存在している必要があります。

operand2

operand2 として、プロパティのフォーマットに対してデータ転送互換を必要とするフォーマットのオペランドを指定します。 詳細については、データ転送の互換性規則の説明を参照してください。

property-name

オブジェクトのプロパティの名前です。

プロパティ名が Natural 識別子構文に対応する場合は、次のように指定できます。

```
create object #o1 of class "Employee"
  #age := #o1.Age
```

プロパティ名がNatural識別子構文に対応しない場合は、次のように山カッコで囲む必要があります。

```
create object #o1 of class "Employee"
  #salary := #o1.<<%Salary>>
```

プロパティ名は、インターフェイス名で修飾することもできます。 同じ名前のプロパティが含まれる複数のインターフェイスがオブジェクトにある場合は、修飾が必要になります。 この場合、修飾したプロパティ名は山カッコで囲む必要があります。

```
create object #o1 of class "Employee"
  #age := #o1.<<PersonalData.Age>>
```

例：

```
define data
  local
  1 #i          (i2)
  1 #o          handle of object
  1 #p          (5) handle of object
  1 #q          (5) handle of object
  1 #salary     (p7.2)
  1 #history    (p7.2/1:10)
  end-define
  * ...
  * Code omitted for brevity.
  * ...
  * Set/Read the Salary property of the object #o.
  #o.Salary := #salary
  #salary := #o.Salary
  * Set/Read the Salary property of
  * the second object of the array #p.
  #p.Salary(2) := #salary
  #salary := #p.Salary(2)
  *
  * Set/Read the SalaryHistory property of the object #o.
  #o.SalaryHistory := #history(1:10)
  #history(1:10) := #o.SalaryHistory
  * Set/Read the SalaryHistory property of
  * the second object of the array #p.
  #p.SalaryHistory(2) := #history(1:10)
  #history(1:10) := #p.SalaryHistory(2)
  *
  * Set the Salary property of each object in #p to the same value.
  #p.Salary(*) := #salary
  * Set the SalaryHistory property of each object in #p
  * to the same value.
  #p.SalaryHistory(*) := #history(1:10)
  *
  * Set the Salary property of each object in #p to the value
  * of the Salary property of the corresponding object in #q.
  #p.Salary(*) := #q.Salary(*)
  * Set the SalaryHistory property of each object in #p to the value
  * of the SalaryHistory property of the corresponding object in #q.
```

```
  #p.SalaryHistory(*) := #q.SalaryHistory(*)
  *
  end
```

配列が値として正しく含まれるオブジェクトハンドルとプロパティの配列を使用するためには、次の知識が重要です。

オブジェクトハンドルの配列のオカレンスのプロパティは、次のインデックス表記で指定します。

```
#p.Salary(2) := #salary
```

配列が値として含まれるプロパティは、常に全体としてアクセスされます。 したがって、プロパティ名でのインデックス表記は不要です。

```
#o.SalaryHistory := #history(1:10)
```

結果的に、配列が値として含まれるオブジェクトハンドルの配列のオカレンスのプロパティは、次のように指定します。

```
#p.SalaryHistory(2) := #history(1:10)
```

61 Natural 予約キーワード

この章では、Naturalプログラミング言語で予約されているすべてのキーワードのリストを提供します。

重要: 命名の競合を避けるため、Natural 予約キーワードを変数の名前として使用しないことを強くお勧めします。

Natural 予約キーワードのアルファベット順リスト

次のリストは、Natural予約キーワードの概要および全般的な情報のみを示しています。疑問がある場合は、コンパイラの キーワードチェック機能を使用してください。

[A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z]

- A -

ABS
ABSOLUTE
ACCEPT
ACTION
ACTIVATION
AD
ADD
AFTER
AL
ALARM
ALL
ALPHA
ALPHABETICALLY
AND
ANY
APPL
APPLICATION
ARRAY
AS
ASC
ASCENDING
ASSIGN
ASSIGNING
ASYNC
AT
ATN
ATT
ATTRIBUTES

AUTH
AUTHORIZATION
AUTO
AVER
AVG

- B -

BACKOUT
BACKWARD
BASE
BEFORE
BETWEEN
BLOCK
BOT
BOTTOM
BREAK
BROWSE
BUT
BX
BY

- C -

CABINET
CALL
CALLDBPROC
CALLING
CALLNAT
CAP
CAPTIONED
CASE
CC
CD
CDID
CF
CHAR
CHARLENGTH
CHARPOSITION
CHILD
CIPH
CIPHER
CLASS
CLOSE
CLR
COALESCE

CODEPAGE
COMMAND
COMMIT
COMPOSE
COMPRESS
COMPUTE
CONCAT
CONDITION
CONST
CONSTANT
CONTEXT
CONTROL
CONVERSATION
COPIES
COPY
COS
COUNT
COUPLED
CS
CURRENT
CURSOR
CV

- D -

DATA
DATAAREA
DATE
DAY
DAYS
DC
DECIDE
DECIMAL
DEFINE
DEFINITION
DELETE
DELIMITED
DELIMITER
DELIMITERS
DESC
DESCENDING
DIALOG
DIALOG-ID
DIGITS
DIRECTION

DISABLED
DISP
DISPLAY
DISTINCT
DIVIDE
DL
DLOGOFF
DLOGON
DNATIVE
DNRET
DO
DOCUMENT
DOEND
DOWNLOAD
DU
DY
DYNAMIC

- E -

EDITED
EJ
EJECT
ELSE
EM
ENCODED
END
END-ALL
END-BEFORE
END-BREAK
END-BROWSE
END-CLASS
END-DECIDE
END-DEFINE
END-ENDDATA
END-ENDFILE
END-ENDPAGE
END-ERROR
END-FILE
END-FIND
END-FOR
END-FUNCTION
END-HISTOGRAM
ENDHOC
END-IF

END-INTERFACE
END-LOOP
END-METHOD
END-NOREC
END-PARAMETERS
END-PARSE
END-PROCESS
END-PROPERTY
END-PROTOTYPE
END-READ
END-REPEAT
END-RESULT
END-SELECT
END-SORT
END-START
END-SUBROUTINE
END-TOPPAGE
END-WORK
ENDING
ENTER
ENTIRE
ENTR
EQ
EQUAL
ERASE
ERROR
ERRORS
ES
ESCAPE
EVEN
EVENT
EVERY
EXAMINE
EXCEPT
EXISTS
EXIT
EXP
EXPAND
EXPORT
EXTERNAL
EXTRACTING

- F -

FALSE

FC
FETCH
FIELD
FIELDS
FILE
FILL
FILLER
FINAL
FIND
FIRST
FL
FLOAT
FOR
FORM
FORMAT
FORMATTED
FORMATTING
FORMS
FORWARD
FOUND
FRAC
FRAMED
FROM
FS
FULL
FUNCTION
FUNCTIONS

- G -

GC
GE
GEN
GENERATED
GET
GFID
GIVE
GIVING
GLOBAL
GLOBALS
GREATER
GT
GUI

- H -

HANDLE
HAVING
HC
HD
HE
HEADER
HEX
HISTOGRAM
HOLD
HORIZ
HORIZONTALLY
HOUR
HOURS
HW

- I -

IA
IC
ID
IDENTICAL
IF
IGNORE
IM
IMMEDIATE
IMPORT
IN
INC
INCCONT
INCDIC
INCDIR
INCLUDE
INCLUDED
INCLUDING
INCMAC
INDEPENDENT
INDEX
INDEXED
INDICATOR
INIT
INITIAL
INNER
INPUT
INSENSITIVE

INSERT
INT
INTEGER
INTERCEPTED
INTERFACE
INTERFACE4
INTERMEDIATE
INTERSECT
INTO
INVERTED
INVESTIGATE
IP
IS
ISN

- J -

JOIN
JUST
JUSTIFIED

- K -

KD
KEEP
KEY
KEYS

- L -

LANGUAGE
LAST
LC
LE
LEAVE
LEAVING
LEFT
LENGTH
LESS
LEVEL
LIB
LIBPW
LIBRARY
LIBRARY-PASSWORD
LIKE
LIMIT

LINDICATOR
LINES
LISTED
LOCAL
LOCKS
LOG
LOG-LS
LOG-PS
LOGICAL
LOOP
LOWER
LS
LT

- M -

MACROAREA
MAP
MARK
MASK
MAX
MC
MCG
MESSAGES
METHOD
MGID
MICROSECOND
MIN
MINUTE
MODAL
MODIFIED
MODULE
MONTH
MORE
MOVE
MOVING
MP
MS
MT
MULTI-FETCH
MULTIPLY

- N -

NAME
NAMED

NAMESPACE
NATIVE
NAVER
NC
NCOUNT
NE
NEWPAGE
NL
NMIN
NO
NODE
NOHDR
NONE
NORMALIZE
NORMALIZED
NOT
NOTIT
NOTITLE
NULL
NULL-HANDLE
NUMBER
NUMERIC

- O -

OBJECT
OBTAIN
OCCURRENCES
OF
OFF
OFFSET
OLD
ON
ONCE
ONLY
OPEN
OPTIMIZE
OPTIONAL
OPTIONS
OR
ORDER
OUTER
OUTPUT

- P -

PACKAGESET
PAGE
PARAMETER
PARAMETERS
PARENT
PARSE
PASS
PASSW
PASSWORD
PATH
PATTERN
PA1
PA2
PA3
PC
PD
PEN
PERFORM
PF*n*（*n* = 1～9）
PF*nn*（*nn* = 10～99）
PGDN
PGUP
PGM
PHYSICAL
PM
POLICY
POS
POSITION
PREFIX
PRINT
PRINTER
PROCESS
PROCESSING
PROFILE
PROGRAM
PROPERTY
PROTOTYPE
PRTY
PS
PT
PW

- Q -

QUARTER
QUERYNO

- R -

RD
READ
READONLY
REC
RECORD
RECORDS
RECURSIVELY
REDEFINE
REDUCE
REFERENCED
REFERENCING
REINPUT
REJECT
REL
RELATION
RELATIONSHIP
RELEASE
REMAINDER
REPEAT
REPLACE
REPORT
REPORTER
REPOSITION
REQUEST
REQUIRED
RESET
RESETTING
RESIZE
RESPONSE
RESTORE
RESULT
RET
RETAIN
RETAINED
RETRY
RETURN
RETURNS
REVERSED
RG

RIGHT
ROLLBACK
ROUNDED
ROUTINE
ROW
ROWS
RR
RS
RULEVAR
RUN

- S -

SA
SAME
SCAN
SCREEN
SCROLL
SECOND
SELECT
SELECTION
SEND
SENSITIVE
SEPARATE
SEQUENCE
SERVER
SET
SETS
SETTIME
SF
SG
SGN
SHORT
SHOW
SIN
SINGLE
SIZE
SKIP
SL
SM
SOME
SORT
SORTED
SORTKEY
SOUND

SPACE
SPECIFIED
SQL
SQLID
SQRT
STACK
START
STARTING
STATEMENT
STATIC
STATUS
STEP
STOP
STORE
SUBPROGRAM
SUBPROGRAMS
SUBROUTINE
SUBSTR
SUBSTRING
SUBTRACT
SUM
SUPPRESS
SUPPRESSED
SUSPEND
SYMBOL
SYNC
SYSTEM

- T -

TAN
TC
TERMINATE
TEXT
TEXTAREA
TEXTVARIABLE
THAN
THEM
THEN
THRU
TIME
TIMESTAMP
TIMEZONE
TITLE
TO

TOP
TOTAL
TP
TR
TRAILER
TRANSACTION
TRANSFER
TRANSLATE
TREQ
TRUE
TS
TYPE
TYPES

- U -

UC
UNDERLINED
UNION
UNIQUE
UNKNOWN
UNTIL
UPDATE
UPLOAD
UPPER
UR
USED
USER
USING

- V -

VAL
VALUE
VALUES
VARGRAPHIC
VARIABLE
VARIABLES
VERT
VERTICALLY
VIA
VIEW

- W -

WH

WHEN
WHERE
WHILE
WINDOW
WITH
WORK
WRITE
WITH_CTE

- X -

XML

- Y -

YEAR

- Z -

ZD
ZP

Natural 予約キーワードのチェックの実行

一部の Natural キーワードは、変数の名前として使用すると不明瞭になります。 特に、Natural ステートメント（ADD、FIND など）やシステム関数（ABS、SUM、など）を特定するキーワードがこれに該当します。このようなキーワードを変数の名前として使用した場合、CALLNAT や WRITE のようなオプションのオペランドのコンテキストでその変数を使うことはできません。

例：

```
DEFINE DATA LOCAL
1 ADD (A10)
END-DEFINE
CALLNAT 'MYSUB' ADD 4     /* ADD is regarded as ADD statement
END
```

プログラミングオブジェクト内の変数名をこのような Natural 予約キーワードと照らし合わせてチェックするために、次のいずれかの機能を利用できます。

- CMPO プロファイルパラメータまたは NTCMPO パラメータマクロの KCHECK キーワードサブパラメータ
- COMPOPT システムコマンドの KCHECK オプション

次のテーブルに、KCHECK でチェックされる Natural 予約キーワードのリストを示します。

A～D	E～F	G～P	R～S	T～W
A-AVER	EJECT	GET	READ	TAN
ABS	ELSE	HISTOGRAM	REDEFINE	TERMINATE
ACCEPT	END	IF	REDUCE	TOP
ADD	END-ALL	IGNORE	REINPUT	TOTAL
ALL	END-BEFORE	IMPORT	REJECT	TRANSFER
A-MAX	END-BREAK	INCCONT	RELEASE	TRUE
A-MIN	END-BROWSE	INCDIC	REPEAT	UNTIL
A-NAVER	END-DECIDE	INCDIR	REQUEST	UPDATE
A-NCOUNT	END-ENDDATA	INCLUDE	RESET	UPLOAD
A-NMIN	END-DECIDE	INCMAC	RESIZE	VAL
ANY	END-ENDDATA	INPUT	RESTORE	VALUE
ASSIGN	END-ENDFILE	INSERT	RET	VALUES
A-SUM	END-ENDPAGE	INT	RETRY	WASTE
AT	END-ERROR	INVESTIGATE	RETURN	WHEN
ATN	END-FILE	LIMIT	ROLLBACK	WHILE
AVER	END-FIND	LOG	RULEVAR	WITH_CTE
BACKOUT	END-FOR	LOOP	RUN	WRITE
BEFORE	END-HISTOGRAM	MAP	SELECT	
BREAK	ENDHOC	MAX	SEND	
BROWSE	END-IF	MIN	SEPARATE	
CALL	END-LOOP	MOVE	SET	
CALLDBPROC	END-NOREC	MULTIPLY	SETTIME	
CALLNAT	END-PARSE	NAVER	SGN	
CLOSE	END-PROCESS	NCOUNT	SHOW	
COMMIT	END-READ	NEWPAGE	SIN	
COMPOSE	END-REPEAT	NMIN	SKIP	
COMPRESS	END-RESULT	NONE	SORT	
COMPUTE	END-SELECT	NULL-HANDLE	SORTKEY	
COPY	END-SORT	OBTAIN	SQRT	
COS	END-START	OLD	STACK	
COUNT	END-SUBROUTINE	ON	START	
CREATE	END-TOPPAGE	OPEN	STOP	
DECIDE	END-WORK	OPTIONS	STORE	
DEFINE	ENTIRE	PARSE	SUBSTR	
DELETE	ESCAPE	PASSW	SUBSTRING	
DISPLAY	EXAMINE	PERFORM	SUBTRACT	
DIVIDE	EXP	POS	SUM	
DLOGOFF	EXPAND	PRINT	SUSPEND	
DLOGON	EXPORT	PROCESS		
DNATIVE	FALSE			
DO	FETCH			
DOEND	FIND			
DOWNLOAD	FOR			
	FORMAT			
	FRAC			

デフォルトでは、キーワードチェックは実行されません。

62 参照プログラム例

この章では、『*プログラミングガイド*』で参照されている追加のプログラム例を示します。

READ ステートメント

次の例は、「*データベースアクセスのためのステートメント*」セクションで参照されています。

READX03 - READ ステートメント（**LOGICAL** 節を含む）

```
** Example 'READX03': READ (with LOGICAL clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
  2 JOB-TITLE
END-DEFINE
*
LIMIT 8
READ EMPLOY-VIEW LOGICAL BY PERSONNEL-ID
  DISPLAY NOTITLE  *ISN      NAME
                  'PERS-NO'  PERSONNEL-ID
                  'POSITION' JOB-TITLE
END-READ
END
```

プログラム READX03 の出力：

```
   ISN              NAME         PERS-NO          POSITION
----------- -------------------- -------- -------------------------

        204 SCHINDLER            11100102 PROGRAMMIERER
        205 SCHIRM               11100105 SYSTEMPROGRAMMIERER
        206 SCHMITT              11100106 OPERATOR
        207 SCHMIDT              11100107 SEKRETAERIN
        208 SCHNEIDER            11100108 SACHBEARBEITER
        209 SCHNEIDER            11100109 SEKRETAERIN
        210 BUNGERT              11100110 SYSTEMPROGRAMMIERER
        211 THIELE               11100111 SEKRETAERIN
```

FIND ステートメント

次の例は、「データベースアクセスのためのステートメント」セクションで参照されています。

FINDX07 - FIND ステートメント（複数節を含む）

```
** Example 'FINDX07': FIND  (with several clauses)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
FIND EMPLOY-VIEW WITH  PHONETIC-NAME = 'JONES' OR = 'BECKR'
                 AND   CITY          = 'BOSTON' THRU 'NEW YORK'
                 BUT NOT               'CHAPEL HILL'
                 SORTED BY NAME
                 WHERE SALARY (1)    < 28000
  DISPLAY NOTITLE NAME FIRST-NAME CITY SALARY (1)
END-FIND
END
```

プログラム FINDX07 の出力：

```
        NAME              FIRST-NAME              CITY           ANNUAL
                                                                 SALARY
-------------------- -------------------- -------------------- ----------

BAKER                PAULINE              DERBY                      4450
JONES                MARTHA               KALAMAZOO                 21000
JONES                KEVIN                DERBY                      7000
```

FINDX08 - FIND ステートメント（**LIMIT** を含む）

```
** Example 'FINDX08': FIND (with LIMIT)
**                     Demonstrates FIND statement with LIMIT option to
**                     terminate program with an error message if the
**                     number of records selected exceeds a specified
**                     limit (no output).
************************************************************************
DEFINE DATA LOCAL
```

```
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
FIND EMPLOY-VIEW WITH LIMIT (5) JOB-TITLE = 'SALES PERSON'
  DISPLAY NAME JOB-TITLE
END-FIND
END
```

プログラム FINDX08 によって発生するランタイムエラー：

```
NAT1005 検索制限の指定より多くのレコードが見つかりました。
```

FINDX09 - FIND ステートメント（***NUMBER**、***COUNTER**、***ISN** を使用）

```
** Example 'FINDX09': FIND (using *NUMBER, *COUNTER, *ISN)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 DEPT
  2 NAME
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY  = 'BOSTON'
                 WHERE DEPT = 'TECH00' THRU 'TECH10'
  DISPLAY NOTITLE
          'COUNTER' *COUNTER NAME DEPT 'ISN' *ISN
  AT START OF DATA
    WRITE '(TOTAL NUMBER IN BOSTON:' *NUMBER ')' /
  END-START
END-FIND
END
```

プログラム FINDX09 の出力：

```
  COUNTER           NAME         DEPARTMENT     ISN
                                    CODE
----------- -------------------- ---------- -----------

(TOTAL NUMBER IN BOSTON:           7 )

          1 STANWOOD             TECH10             782
          2 PERREAULT            TECH10             842
```

FINDX10 - FIND ステートメント（**READ** との組み合わせ）

```
** Example 'FINDX10': FIND (combined with READ)
************************************************************************
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 FINDX10 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 ***
```

FINDX11 - FIND NUMBER ステートメント（***NUMBER** を含む）

```
** Example 'FINDX11': FIND NUMBER (with *NUMBER)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 FIRST-NAME
  2 NAME
  2 CITY
  2 JOB-TITLE
  2 SALARY        (1)
*
1 #PERCENT        (N.2)
1 REDEFINE #PERCENT
  2 #WHOLE-NBR    (N2)
1 #ALL-BOST       (N3.2)
1 #SECR-ONLY      (N3.2)
1 #BOST-NBR       (N3)
1 #SECR-NBR       (N3)
END-DEFINE
*
F1. FIND NUMBER EMPLOY-VIEW WITH CITY = 'BOSTON'
F2. FIND NUMBER EMPLOY-VIEW WITH CITY = 'BOSTON'
                            AND JOB-TITLE = 'SECRETARY'
*
MOVE *NUMBER(F1.) TO #ALL-BOST  #BOST-NBR
MOVE *NUMBER(F2.) TO #SECR-ONLY #SECR-NBR
DIVIDE #ALL-BOST INTO #SECR-ONLY GIVING #PERCENT
*
WRITE TITLE LEFT JUSTIFIED UNDERLINED
  'There are' #BOST-NBR 'employees in the Boston offices.' /
  #SECR-NBR '(=' #WHOLE-NBR (EM=99%')') 'are secretaries.'
*
SKIP 1
FIND EMPLOY-VIEW WITH CITY      = 'BOSTON'
                 AND  JOB-TITLE = 'SECRETARY'
  DISPLAY NAME FIRST-NAME JOB-TITLE SALARY (1)
END-FIND
END
```

プログラム FINDX11 の出力：

```
There are    7 employees in the Boston offices.
   3 (= 42%) are secretaries.
------------------------------------------------------------------------------

        NAME              FIRST-NAME              CURRENT             ANNUAL
                                                  POSITION            SALARY
-------------------- -------------------- ------------------------ ----------

SHAW                 LESLIE               SECRETARY                     18000
CREMER               WALT                 SECRETARY                     20000
COHEN                JOHN                 SECRETARY                     16000
```

READ および FIND ステートメントのネスト

次の例は、「*データベース処理ループ*」セクションで参照されています。

READX04 - READ ステートメント（**FIND** およびシステム変数 ***NUMBER** と ***COUNTER** との組み合わせ）

```
** Example 'READX04': READ (in combination with FIND and the system
**                     variables *NUMBER and *COUNTER)
************************************************************************
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 10
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  FD. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      ENTER
    END-NOREC
    /*
    DISPLAY NOTITLE
            *COUNTER (RD.)(NL=8) NAME         (AL=15) FIRST-NAME (AL=10)
            *NUMBER  (FD.)(NL=8) *COUNTER (FD.)(NL=8) MAKE
  END-FIND
END-READ
END
```

プログラム READX04 の出力：

```
  CNT          NAME       FIRST-NAME   NMBR      CNT             MAKE
--------- --------------- ---------- --------- --------- --------------------

       1 JONES           VIRGINIA           1         1 CHRYSLER
       2 JONES           MARSHA             2         1 CHRYSLER
       2 JONES           MARSHA             2         2 CHRYSLER
       3 JONES           ROBERT             1         1 GENERAL MOTORS
       4 JONES           LILLY              2         1 FORD
       4 JONES           LILLY              2         2 MG
       5 JONES           EDWARD             1         1 GENERAL MOTORS
       6 JONES           MARTHA             1         1 GENERAL MOTORS
       7 JONES           LAUREL             1         1 GENERAL MOTORS
       8 JONES           KEVIN              1         1 DATSUN
       9 JONES           GREGORY            1         1 FORD
      10 JOPER           MANFRED            0         0
```

LIMITX01 - LIMIT ステートメント（**READ**、**FIND** ループ処理）

```
** Example 'LIMITX01': LIMIT  (for READ, FIND loop processing)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 4
*
READ EMPLOY-VIEW BY NAME STARTING FROM 'A'
  FIND VEH-VIEW WITH PERSONNEL-ID = EMPLOY-VIEW.PERSONNEL-ID
    IF NO RECORDS FOUND
      MOVE 'NO CAR' TO MAKE
    END-NOREC
    DISPLAY PERSONNEL-ID NAME FIRST-NAME MAKE
  END-FIND
END-READ
END
```

プログラム LIMITX01 の出力：

```
Page      1                                                  04-12-13  14:01:57

PERSONNEL-ID         NAME              FIRST-NAME              MAKE
------------ -------------------- -------------------- --------------------

             ABELLAN              KEPA                 NO CAR
30000231     ACHIESON             ROBERT               FORD
             ADAM                 SIMONE               NO CAR
20008800     ADKINSON             JEFF                 GENERAL MOTORS
```

ACCEPT および REJECT ステートメント

次の例は、「*ACCEPT/REJECT を使用したレコードの選択*」セクションで参照されています。

ACCEPX04 - ACCEPT IF ... LESS THAN ... ステートメント

```
** Example 'ACCEPX04': ACCEPT IF ... LES THAN ...
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY (1)
END-DEFINE
*
LIMIT 20
READ EMPLOY-VIEW BY PERSONNEL-ID = '20017000'
  ACCEPT IF SALARY (1) LESS THAN 38000
  DISPLAY NOTITLE PERSONNEL-ID NAME JOB-TITLE SALARY (1)
END-READ
END
```

プログラム ACCEPX04 の出力：

```
PERSONNEL         NAME                 CURRENT            ANNUAL
   ID                                  POSITION           SALARY
--------- -------------------- ------------------------- ----------

20017000  CREMER               ANALYST                        34000
20017100  MARKUSH              TRAINEE                        22000
20017400  NEEDHAM              PROGRAMMER                     32500
20017500  JACKSON              PROGRAMMER                     33000
20017600  PIETSCH              SECRETARY                      22000
20017700  PAUL                 SECRETARY                      23000
```

```
20018000  FARRIS               PROGRAMMER                     30500
20018100  EVANS                PROGRAMMER                     31000
20018200  HERZOG               PROGRAMMER                     31500
20018300  LORIE                SALES PERSON                   28000
20018400  HALL                 SALES PERSON                   30000
20018500  JACKSON              MANAGER                        36000
20018800  SMITH                SECRETARY                      24000
20018900  LOWRY                SECRETARY                      25000
```

ACCEPX05 - ACCEPT IF ... AND ... ステートメント

```
** Example 'ACCEPX05': ACCEPT IF ... AND ...
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 JOB-TITLE
  2 SALARY (1:2)
END-DEFINE
*
LIMIT 6
READ EMPLOY-VIEW PHYSICAL WHERE SALARY(2) > 0
  ACCEPT IF  SALARY(1) > 10000
         AND SALARY(1) < 50000
  DISPLAY (AL=15) 'SALARY I' SALARY (1) 'SALARY II'  SALARY (2)
                  NAME JOB-TITLE  CITY
END-READ
END
```

プログラム ACCEPX05 の出力：

```
Page      1                                              04-12-13  14:05:28

 SALARY I  SALARY II       NAME           CURRENT          CITY
                                         POSITION
---------- ---------- --------------- --------------- ---------------

     48000      46000 SPENGLER        SACHBEARBEITER  DARMSTADT
     45000      40000 SPECK           SACHBEARBEITER  DARMSTADT
     48000      46000 SCHINDLER       PROGRAMMIERER   HEPPENHEIM
     36000      32000 SCHMIDT         SEKRETAERIN     HEPPENHEIM
```

ACCEPX06 - REJECT IF ... OR ... ステートメント

```
** Example 'ACCEPX06': REJECT IF ... OR ...
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 SALARY (1)
  2 JOB-TITLE
  2 CITY
  2 NAME
END-DEFINE
*
LIMIT 20
READ EMPLOY-VIEW LOGICAL BY PERSONNEL-ID = '20017000'
  REJECT IF SALARY (1) < 20000
         OR SALARY (1) > 26000
  DISPLAY NOTITLE SALARY (1) NAME JOB-TITLE CITY
END-READ
END
```

プログラム ACCEPX06 の出力：

```
  ANNUAL           NAME                  CURRENT                   CITY
  SALARY                                 POSITION
---------- -------------------- ------------------------- --------------------

     22000 MARKUSH              TRAINEE                   LOS ANGELES
     22000 PIETSCH              SECRETARY                 VISTA
     23000 PAUL                 SECRETARY                 NORFOLK
     24000 SMITH                SECRETARY                 SILVER SPRING
     25000 LOWRY                SECRETARY                 LEXINGTON
```

AT START OF DATA および AT END OF DATA ステートメント

次の例は、「*AT START/END OF DATA ステートメント*」セクションで参照されています。

ATENDX01 - AT END OF DATA ステートメント

```
** Example 'ATENDX01': AT END OF DATA
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
READ (6) EMPLOY-VIEW BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE NAME JOB-TITLE
  AT END OF DATA
    WRITE / 'LAST PERSON SELECTED:' OLD(NAME)
  END-ENDDATA
END-READ
END
```

プログラム ATENDX01 の出力：

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

CREMER               ANALYST
MARKUSH              TRAINEE
GEE                  MANAGER
KUNEY                DBA
NEEDHAM              PROGRAMMER
JACKSON              PROGRAMMER

LAST PERSON SELECTED: JACKSON
```

ATSTAX02 - AT START OF DATA ステートメント

```
** Example 'ATSTAX02': AT START OF DATA
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 SALARY    (1)
  2 CURR-CODE (1)
  2 BONUS     (1,1)
END-DEFINE
*
```

```
LIMIT 3
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
  DISPLAY NAME FIRST-NAME SALARY(1) BONUS(1,1) CURR-CODE (1)
  /*
  AT START OF DATA
    WRITE NOTITLE *DAT4E /
  END-START
END-FIND
END
```

プログラム ATSTAX02 の出力：

```
        NAME              FIRST-NAME        ANNUAL     BONUS    CURRENCY
                                            SALARY               CODE
-------------------- -------------------- ---------- ---------- --------

13/12/2004

DE JUAN              JAVIER                1988000          0 PTA
DE LA MADRID         ANSELMO               3120000          0 PTA
PINERO               PAULA                 1756000          0 PTA
```

WRITEX09 - WRITE ステートメント（**AT END OF DATA** との組み合わせ）

```
** Example 'WRITEX09': WRITE (in combination with AT END OF DATA )
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 BIRTH
  2 JOB-TITLE
  2 DEPT
END-DEFINE
*
READ (3) EMPLOY-VIEW BY CITY
  DISPLAY NOTITLE NAME BIRTH (EM=YYYY-MM-DD) JOB-TITLE
  WRITE 38T 'DEPT CODE:' DEPT
  /*
  AT END OF DATA
    WRITE / 'LAST PERSON SELECTED:' OLD(NAME)
  END-ENDDATA
  SKIP 1
END-READ
END
```

プログラム WRITEX09 の出力：

```
        NAME            DATE             CURRENT
                         OF             POSITION
                       BIRTH
-------------------- ---------- -------------------------

SENKO                1971-09-11 PROGRAMMER
                                      DEPT CODE: TECH10

GODEFROY             1949-01-09 COMPTABLE
                                      DEPT CODE: COMP02

CANALE               1942-01-01 CONSULTANT
                                      DEPT CODE: TECH03


LAST PERSON SELECTED: CANALE
```

DISPLAY および WRITE ステートメント

次の例は、「*DISPLAY および WRITE ステートメント*」セクションで参照されています。

DISPLX13 - DISPLAY ステートメント（**WRITE** を使用する **WRITEX08** と比較）

```
** Example 'DISPLX13': DISPLAY (compare with WRITEX08 using WRITE)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 SALARY (2)
  2 BONUS  (1,1)
  2 CITY
END-DEFINE
*
LIMIT 2
READ EMPLOY-VIEW WITH CITY = 'CHAPEL HILL' WHERE BONUS(1,1) NE 0
  /*
  DISPLAY 'PERS/ID' PERSONNEL-ID  NAME / FIRST-NAME
          '**' '=' SALARY(1:2) 'BONUS' BONUS(1,1) CITY (AL=15)
  /*
  SKIP 1
END-READ
END
```

プログラム DISPLX13 の出力：

```
Page      1                                                  04-12-13  14:11:28

  PERS           NAME             ANNUAL       BONUS         CITY
   ID         FIRST-NAME          SALARY
-------- -------------------- ------------ ---------- ---------------

20027000 CUMMINGS             **      41000       1500 CHAPEL HILL
         PUALA                        38900

20000200 WOOLSEY              **      26000       3000 CHAPEL HILL
         LOUISE                       24700
```

WRITEX08 - WRITE ステートメント（**DISPLAY** を使用する **DISPLX13** と比較）

```
** Example 'WRITEX08': WRITE  (compare with DISPLX13 using DISPLAY)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 SALARY (2)
  2 BONUS  (1,1)
  2 CITY
END-DEFINE
*
LIMIT 2
READ EMPLOY-VIEW WITH CITY = 'CHAPEL HILL' WHERE BONUS(1,1) NE 0
  /*
  WRITE 'PERS/ID' PERSONNEL-ID  NAME / FIRST-NAME
        '**' '=' SALARY(1:2) 'BONUS' BONUS(1,1) CITY (AL=15)
  /*
  SKIP 1
END-READ
END
```

プログラム WRITEX08 の出力：

```
Page      1                                                  04-12-13  14:12:43

PERS/ID 20027000 CUMMINGS
PUALA                ** ANNUAL SALARY:      41000      38900 BONUS       1500
CHAPEL HILL

PERS/ID 20000200 WOOLSEY
LOUISE               ** ANNUAL SALARY:      26000      24700 BONUS       3000
CHAPEL HILL
```

DISPLX14 - DISPLAY ステートメント（**AL**、**SF**、*n***X** を含む）

```
** Example 'DISPLX14': DISPLAY  (with AL, SF and nX)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 FIRST-NAME
  2 NAME
  2 ADDRESS-LINE (1)
  2 TELEPHONE
    3 AREA-CODE
    3 PHONE
  2 CITY
END-DEFINE
*
READ (3) EMPLOY-VIEW BY NAME STARTING FROM 'W'
  DISPLAY (AL=15 SF=5) NAME CITY / ADDRESS-LINE(1) 2X TELEPHONE
  SKIP 1
END-READ
END
```

プログラム DISPLX14 の出力：

```
Page      1                                                04-12-13  14:14:00

     NAME                CITY                      TELEPHONE
                        ADDRESS
                                                AREA              TELEPHONE
                                                CODE
---------------     ---------------  ---------------     ---------------

WABER               HEIDELBERG       06221               456452
                    ERBACHERSTR. 78

WADSWORTH           DERBY            0332                515365
                    56 PINECROFT CO

WAGENBACH           FRANKFURT        069                 983218
                    BECKERSTR. 4
```

WRITEX09 - WRITE ステートメント（**AT END OF DATA** との組み合わせ）

```
** Example 'WRITEX09': WRITE (in combination with AT END OF DATA )
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 BIRTH
  2 JOB-TITLE
  2 DEPT
END-DEFINE
*
READ (3) EMPLOY-VIEW BY CITY
  DISPLAY NOTITLE NAME BIRTH (EM=YYYY-MM-DD) JOB-TITLE
  WRITE 38T 'DEPT CODE:' DEPT
  /*
  AT END OF DATA
    WRITE / 'LAST PERSON SELECTED:' OLD(NAME)
  END-ENDDATA
  SKIP 1
END-READ
END
```

プログラム WRITEX09 の出力：

```
        NAME            DATE             CURRENT
                         OF             POSITION
                       BIRTH
-------------------- ---------- -------------------------

SENKO                1971-09-11 PROGRAMMER
                                      DEPT CODE: TECH10

GODEFROY             1949-01-09 COMPTABLE
                                      DEPT CODE: COMP02

CANALE               1942-01-01 CONSULTANT
                                      DEPT CODE: TECH03


LAST PERSON SELECTED: CANALE
```

DISPLAY ステートメント

次の例は、「*ページタイトル、改ページ、空行*」セクションで参照されています。

DISPLX21 DISPLAY ステートメント（スラッシュ '/' を含む。**WRITE** と比較）

```
** Example 'DISPLX21': DISPLAY (usage of slash '/' in DISPLAY and WRITE)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 ADDRESS-LINE (1)
END-DEFINE
*
WRITE TITLE LEFT JUSTIFIED UNDERLINED
       *TIME
   5X  'PEOPLE LIVING IN SALT LAKE CITY'
   21X 'PAGE:' *PAGE-NUMBER /
   15X 'AS OF' *DAT4E //
*
WRITE TRAILER UNDERLINED 'REGISTER OF' / 'SALT LAKE CITY'
*
READ (2) EMPLOY-VIEW WITH CITY = 'SALT LAKE CITY'
  DISPLAY  NAME /
           FIRST-NAME
           'HOME/CITY' CITY
           'STREET/OR BOX NO.' ADDRESS-LINE (1)
  SKIP 1
END-READ
END
```

プログラム DISPLX21 の出力：

```
14:15:50.1     PEOPLE LIVING IN SALT LAKE CITY                    PAGE:      1
               AS OF 13/12/2004


------------------------------------------------------------------------------
        NAME                 HOME                 STREET
     FIRST-NAME              CITY               OR BOX NO.
-------------------- -------------------- --------------------

ANDERSON             SALT LAKE CITY       3701 S. GEORGE MASON
JENNY
```

```
SAMUELSON            SALT LAKE CITY       7610 W. 86TH STREET
MARTIN

                                    REGISTER OF
                                  SALT LAKE CITY
------------------------------------------------------------------------------
```

列ヘッダー

次の例は、「*列ヘッダー*」セクションで参照されています。

DISPLX15 - DISPLAY ステートメント（**FC**、**UC** を含む）

```
** Example 'DISPLX15': DISPLAY  (with FC, UC)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 FIRST-NAME
  2 NAME
  2 ADDRESS-LINE (1)
  2 CITY
  2 TELEPHONE
    3 AREA-CODE
    3 PHONE
END-DEFINE
*
FORMAT AL=12 GC== UC=%
*
READ (3) EMPLOY-VIEW BY NAME STARTING FROM 'R'
  DISPLAY NOTITLE (FC=*)
          NAME FIRST-NAME CITY  (FC=- UC=-) /
          ADDRESS-LINE(1) TELEPHONE
  SKIP 1
END-READ
END
```

プログラム DISPLX15 の出力：

```
****NAME**** *FIRST-NAME* ----CITY---- ========TELEPHONE========
                          **ADDRESS***
                                        ****AREA**** *TELEPHONE**
                                        ****CODE****
%%%%%%%%%%%% %%%%%%%%%%%% ------------ %%%%%%%%%%%% %%%%%%%%%%%%

RACKMANN     MARIAN       FRANKFURT    069          375849
                          FINKENSTR. 1
```

```
RAMAMOORTHY  TY           SEPULVEDA    209          175-1885
                          12018 BROOKS

RAMAMOORTHY  TIMMIE       SEATTLE      206          151-4673
                          921-178TH PL
```

DISPLX16 - DISPLAY ステートメント（'/'、'*text*'、'*text*/*text*' を含む）

```
** Example 'DISPLX16': DISPLAY  (with '/', 'text', 'text/text')
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 FIRST-NAME
  2 NAME
  2 ADDRESS-LINE (1)
  2 CITY
  2 TELEPHONE
    3 AREA-CODE
    3 PHONE
END-DEFINE
*
READ (5) EMPLOY-VIEW BY NAME STARTING FROM 'E'
  DISPLAY NOTITLE
    '/'          NAME       (AL=12) /* suppressed header
    'FIRST/NAME' FIRST-NAME (AL=10) /* two-line user-defined header
    'ADDRESS'    CITY /             /* user-defined header
    ' '          ADDRESS-LINE(1)    /* 'blank' header
                 TELEPHONE (HC=L)   /* default header
  SKIP 1
END-READ
END
```

プログラム DISPLX16 の出力：

```
               FIRST          ADDRESS        TELEPHONE
                NAME
                                              AREA   TELEPHONE
                                              CODE
             ---------- -------------------- ------ ---------------

EAVES        TREVOR     DERBY                0332   657623
                        17 HARTON ROAD

ECKERT       KARL       OBERRAMSTADT         06154  99722
                        FORSTWEG 22

ECKHARDT     RICHARD    DARMSTADT
                        BRESLAUERPL. 4
```

```
EDMUNDSON    LES        TULSA                918    945-4916
                        2415 ALSOP CT.

EGGERT       HERMANN    STUTTGART            0711   981237
                        RABENGASSE 8
```

フィールド出力関連パラメータ

次の例は、「*フィールドの出力に影響を与えるパラメータ*」セクションで参照されています。

これらの例は、LC、IC、TC、AL、NL、IS、ZP、ES の各パラメータ、および SUSPEND IDENTICAL SUPPRESS ステートメントの使用法を示すために提供されています。

DISPLX17 - DISPLAY ステートメント（**NL**、**AL**、**IC**、**LC**、**TC** を含む）

```
** Example 'DISPLX17': DISPLAY (with NL, AL, IC, LC, TC)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 FIRST-NAME
  2 NAME
  2 SALARY (1)
  2 BONUS  (1,1)
END-DEFINE
*
READ (3) EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  DISPLAY NOTITLE (IS=ON NL=15)
                    NAME
    '-' '='         FIRST-NAME (AL=12)
    'ANNUAL SALARY' SALARY(1)  (LC=USD   TC=.00)   /
    '+ BONUSES'     BONUS(1,1) (IC='+  ' TC=.00)
  SKIP 1
END-READ
END
```

プログラム DISPLX17 の出力：

```
        NAME           FIRST-NAME       ANNUAL SALARY
                                          + BONUSES
-------------------- -------------- ----------------------

JONES                - VIRGINIA     USD           46000.00
                                                +  9000.00

                     - MARSHA       USD           50000.00
```

```
                                                 +  0.00

                    - ROBERT       USD           31000.00
                                                 +  0.00
```

DISPLX18 - DISPLAY ステートメント（**SF**、**AL**、**UC**、**LC**、**IC**、**TC** のデフォルト設定を使用。**DISPLX19** と比較）

```
** Example 'DISPLX18': DISPLAY  (using default settings for SF, AL, UC,
**                       LC, IC, TC and compare with DISPLX19)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 SALARY    (1)
  2 BONUS     (1,1)
END-DEFINE
*
FIND (6) EMPLOY-VIEW WITH CITY = 'CHAPEL HILL'
  DISPLAY NAME FIRST-NAME SALARY(1) BONUS(1,1)
END-FIND
END
```

プログラム DISPLX18 の出力：

```
Page      1                                                 04-12-13  14:20:48

        NAME              FIRST-NAME        ANNUAL     BONUS
                                            SALARY
------------------- -------------------- ---------- ----------

KESSLER             CLARE                     41000          0
ADKINSON            DAVID                     24000          0
GEE                 TOMMIE                    39500          0
HERZOG              JOHN                      31500          0
QUILLION            TIMOTHY                   30500          0
CUMMINGS            PUALA                     41000       1500
```

DISPLX19 - DISPLAY ステートメント（**SF**、**AL**、**LC**、**IC**、**TC** を含む。**DISPLX18** と比較）

```
** Example 'DISPLX19': DISPLAY  (with SF, AL, LC, IC, TC and compare
**                      with DISPLX19)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 SALARY (1)
  2 BONUS  (1,1)
END-DEFINE
*
FORMAT SF=3 AL=15 UC==
*
FIND (6) EMPLOY-VIEW WITH CITY = 'CHAPEL HILL'
  DISPLAY (NL=10)
    NAME
    FIRST-NAME  (LC='- ' UC=-)
    SALARY (1)  (LC=USD)
    BONUS (1,1) (IC='*** ' TC=' ***')
END-FIND
END
```

プログラム DISPLX19 の出力：

```
Page      1                                              04-12-13  14:21:57

     NAME            FIRST-NAME          ANNUAL              BONUS
                                         SALARY
===============   -----------------   ==============   ====================

KESSLER           - CLARE             USD      41000             *** 0 ***
ADKINSON          - DAVID             USD      24000             *** 0 ***
GEE               - TOMMIE            USD      39500             *** 0 ***
HERZOG            - JOHN              USD      31500             *** 0 ***
QUILLION          - TIMOTHY           USD      30500             *** 0 ***
CUMMINGS          - PUALA             USD      41000          *** 1500 ***
```

SUSPEX01 - SUSPEND IDENTICAL SUPPRESS ステートメント（**DISPLAY** のパラメータ **IS**、**ES**、**ZP** との組み合わせ）

```
** Example 'SUSPEX01': SUSPEND IDENTICAL SUPPRESS (in conjunction with
**                      parameters IS, ES, ZP in DISPLAY)
************************************************************************
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 '******' TO MAKE
    END-NOREC
    DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
            NAME       (RD.)
            FIRST-NAME (RD.)
            MAKE       (FD.) (IS=OFF)
  END-FIND
END-READ
END
```

プログラム SUSPEX01 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
JOPER           MANFRED         ******
```

```
JOUSSELIN       DANIEL          RENAULT
JUBE            GABRIEL         ******
JUNG            ERNST           ******
JUNKIN          JEREMY          ******
KAISER          REINER          ******
```

SUSPEX02 - SUSPEND IDENTICAL SUPPRESS ステートメント（**DISPLAY** のパラメータ **IS**、**ES**、**ZP** との組み合わせ）**IS=OFF** である以外は **SUSPEX01** と同じ

```
** Example 'SUSPEX02': SUSPEND IDENTICAL SUPPRESS (in conjunction with
**                     parameters IS, ES, ZP in DISPLAY)
**                     Identical to SUSPEX01, but with IS=OFF.
************************************************************************
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 '******' TO MAKE
    END-NOREC
    DISPLAY NOTITLE (ES=OFF IS=OFF ZP=ON AL=15)
            NAME       (RD.)
            FIRST-NAME (RD.)
            MAKE       (FD.) (IS=OFF)
  END-FIND
END-READ
END
```

プログラム SUSPEX02 の出力：

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

JONES           VIRGINIA        CHRYSLER
JONES           MARSHA          CHRYSLER
JONES           MARSHA          CHRYSLER
JONES           ROBERT          GENERAL MOTORS
JONES           LILLY           FORD
```

```
JONES           LILLY           MG
JONES           EDWARD          GENERAL MOTORS
JONES           MARTHA          GENERAL MOTORS
JONES           LAUREL          GENERAL MOTORS
JONES           KEVIN           DATSUN
JONES           GREGORY         FORD
JOPER           MANFRED         ******
JOUSSELIN       DANIEL          RENAULT
JUBE            GABRIEL         ******
JUNG            ERNST           ******
JUNKIN          JEREMY          ******
KAISER          REINER          ******
```

COMPRX03 - COMPRESS ステートメント

```
** Example 'COMPRX03': COMPRESS (using parameters LC and TC)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 SALARY     (1)
  2 CURR-CODE  (1)
  2 LEAVE-DUE
  2 NAME
  2 FIRST-NAME
  2 JOB-TITLE
*
1 #SALARY      (N9)
1 #FULL-SALARY (A25)
1 #VACATION    (A11)
END-DEFINE
*
READ (3) EMPLOY-VIEW WITH CITY = 'BOSTON'
  MOVE SALARY(1) TO #SALARY
  COMPRESS 'SALARY  :' CURR-CODE(1) #SALARY INTO #FULL-SALARY
  COMPRESS 'VACATION:' LEAVE-DUE            INTO #VACATION
  /*
  DISPLAY NOTITLE NAME FIRST-NAME
           'JOB DESCRIPTION' JOB-TITLE  (LC='JOB     : ')  /
           '/'               #FULL-SALARY                   /
           '/'               #VACATION  (TC='DAYS')
  SKIP 1
END-READ
END
```

プログラム COMPRX03 の出力：

```
        NAME              FIRST-NAME               JOB DESCRIPTION
-------------------- -------------------- ----------------------------------

SHAW                 LESLIE               JOB     : SECRETARY
                                          SALARY  : USD 18000
                                          VACATION: 2DAYS

STANWOOD             VERNON               JOB     : PROGRAMMER
                                          SALARY  : USD 31000
                                          VACATION: 1DAYS

CREMER               WALT                 JOB     : SECRETARY
                                          SALARY  : USD 20000
                                          VACATION: 3DAYS
```

編集マスク

次の例は、「*編集マスク - EM パラメータ*」セクションで参照されています。

EDITMX03 - 編集マスク（英数字フィールドの異なる **EM**）

```
** Example 'EDITMX03': Edit mask (different EM for alpha-numeric fields)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
  2 SALARY(1)
END-DEFINE
*
LIMIT 3
READ EMPLOY-VIEW BY PERSONNEL-ID FROM '20018000'
                 WHERE SALARY(1) = 28000 THRU 30000
  DISPLAY 'N A M E'  NAME       (EM=X^^X^^X^^X^^X^^X^^X^^X^^X^^X) /
          'NAME HEX' NAME       (EM=H^H^H^H^H^H^H^H^H^H^H)
                     FIRST-NAME (EM=' - 'X(15)*)
                     CITY       (EM=X..X(10))
  SKIP 1
END-READ
END
```

プログラム EDITMX03 の出力：

```
Page      1                                                  04-12-13  14:26:57

            N A M E                  FIRST-NAME         CITY
            NAME HEX
------------------------------ ------------------ -------------

L  O  R  I  E                     - JEAN-PAUL      * C..LEVELAND
D3 D6 D9 C9 C5 40 40 40 40 40 40

H  A  L  L                        - ARTHUR         * A..NN ARBER
C8 C1 D3 D3 40 40 40 40 40 40 40

V  A  S  W  A  N  I               - TOMMIE         * M..ONTERREY
E5 C1 E2 E6 C1 D5 C9 40 40 40 40
```

EDITMX04 - 編集マスク（数値フィールドの異なる **EM**）

```
** Example 'EDITMX04': Edit mask (different EM for numeric fields)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 SALARY (1)
  2 BONUS  (1,1)
  2 LEAVE-DUE
END-DEFINE
*
LIMIT 2
READ EMPLOY-VIEW BY PERSONNEL-ID = '20018000'
                 WHERE SALARY(1) = 28000 THRU 30000
  DISPLAY (SF=4)
          'N A M E'    NAME
          'SALARY'     SALARY(1)  (EM=*USD^ZZZ,999)
          'BONUS (ZZ)' BONUS(1,1) (EM=S*ZZZ,999) /
          'BONUS (Z9)' BONUS(1,1) (EM=SZ99,999+) /
          '->' '='     BONUS(1,1) (EM=-999,999)
          'VAC/DUE'    LEAVE-DUE  (EM=+999)
  SKIP 1
END-READ
END
```

プログラム EDITMX04 の出力：

```
Page      1                                                 04-12-13  14:27:43

      N A M E             SALARY       BONUS (ZZ)     VAC
                                       BONUS (Z9)     DUE
                                          BONUS
--------------------    -----------    -----------    ---

LORIE                   USD *28,000    +**4,000       +13
                                       + 04,000+
                                       ->  004,000

HALL                    USD *30,000    +**5,000       +14
                                       + 05,000+
                                       ->  005,000
```

EDITMX05 - 編集マスク（日付および時刻システム変数の **EM**）

```
** Example 'EDITMX05': Edit mask (EM for date and time system variables)
************************************************************************
WRITE NOTITLE //
  'DATE INTERNAL :' *DATX (DF=L) /
  '              :' *DATX (EM=N(9)' 'ZW.'WEEK 'YYYY)  /
  '              :' *DATX (EM=ZZJ'.DAY 'YYYY)        /
  '     ROMAN    :' *DATX (EM=R)  /
  '     AMERICAN :' *DATX (EM=MM/DD/YYYY)      12X 'OR  ' *DAT4U /
  '     JULIAN   :' *DATX (EM=YYYYJJJ)         15X 'OR  ' *DAT4J /
  '     GREGORIAN:' *DATX (EM=ZD.''L(10)''YYYY) 5X 'OR  ' *DATG ///
  'TIME INTERNAL :' *TIMX                      14X 'OR  ' *TIME /
  '              :' *TIMX (EM=HH.II.SS.T) /
  '              :' *TIMX (EM=HH.II.SS' 'AP) /
  '              :' *TIMX (EM=HH)
END
```

プログラム EDITMX05 の出力：

```


DATE INTERNAL : 2004-12-13
              : Monday 51.WEEK 2004
              : 348.DAY 2004
     ROMAN    : MMIV
     AMERICAN : 12/13/2004           OR   12/13/2004
     JULIAN   : 2004348              OR   2004348
     GREGORIAN: 13.December2004      OR   13December 2004


TIME INTERNAL : 14:28:49             OR   14:28:49.1
              : 14.28.49.1
```

```
              : 02.28.49 PM
              : 14
```

WRITE ステートメントを含む DISPLAY VERT

WRITEX10 - WRITE ステートメント（*n***T**、**T*****field*、**P*****field* を含む）

```
** Example 'WRITEX10': WRITE (with nT, T*field and P*field)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 JOB-TITLE
  2 NAME
  2 SALARY (1)
  2 BONUS  (1,1)
END-DEFINE
*
READ (3) EMPLOY-VIEW WITH JOB-TITLE FROM 'SALES PERSON'
  DISPLAY NOTITLE NAME 30T JOB-TITLE
          VERT AS 'SALARY/BONUS' SALARY(1) BONUS(1,1)
  AT BREAK OF JOB-TITLE
    WRITE 20T 'AVERAGE' T*JOB-TITLE OLD(JOB-TITLE) (AL=15)
              '(SAL)'   P*SALARY AVER(SALARY(1))  /
          46T '(BON)'   P*BONUS  AVER(BONUS(1,1)) /
  END-BREAK
  SKIP 1
END-READ
END
```

プログラム WRITEX10 の出力：

```
        NAME                          CURRENT            SALARY
                                     POSITION            BONUS
--------------------          ------------------------ ----------

SAMUELSON                     SALES PERSON                  32000
                                                             6000

PAPAYANOPOULOS                SALES PERSON                  34000
                                                             7000

HELL                          SALES PERSON                  38000
                                                             9000

                   AVERAGE    SALES PERSON    (SAL)         34666
                                              (BON)          7333
```

AT BREAK ステートメント

次の例は、「*コントロールブレイク*」セクションで参照されています。

ATBREX06 - AT BREAK OF ステートメント（**NMIN**、**NAVER**、**NCOUNT** を **MIN**、**AVER**、**COUNT** と比較）

```
** Example 'ATBREX06': AT BREAK OF (comparing NMIN, NAVER, NCOUNT with
**                      MIN, AVER, COUNT)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 SALARY (1:2)
END-DEFINE
*
WRITE TITLE '-- SALARY STATISTICS BY CITY --' /
*
READ (2) EMPLOY-VIEW WITH CITY = 'NEW YORK'
  DISPLAY CITY  'SALARY (1)' SALARY(1) 15X 'SALARY (2)' SALARY(2)
  AT BREAK OF CITY
    WRITE /
      14T 'S A L A R Y   (1)'         39T 'S A L A R Y  (2)'          /
      13T '-   MIN:' MIN(SALARY(1))   38T '-   MIN:' MIN(SALARY(2))   /
      13T '-  AVER:' AVER(SALARY(1))  38T '-  AVER:' AVER(SALARY(2))  /
      16T COUNT(SALARY(1)) 'RECORDS'  41T COUNT(SALARY(2)) 'RECORDS'  //
      13T '-  NMIN:' NMIN(SALARY(1))  38T '-  NMIN:' NMIN(SALARY(2))  /
      13T '- NAVER:' NAVER(SALARY(1)) 38T '- NAVER:' NAVER(SALARY(2)) /
      16T NCOUNT(SALARY(1)) 'RECORDS' 41T NCOUNT(SALARY(2)) 'RECORDS'
  END-BREAK
END-READ
END
```

プログラム ATBREX06 の出力：

```
                        -- SALARY STATISTICS BY CITY --

        CITY          SALARY (1)               SALARY (2)
-------------------- ----------               ----------

NEW YORK                   17000                    16100
NEW YORK                   38000                    34900

             S A L A R Y   (1)        S A L A R Y  (2)
            -   MIN:      17000      -   MIN:      16100
            -  AVER:      27500      -  AVER:      25500
                     2 RECORDS                2 RECORDS
```

```
          -  NMIN:      17000      -  NMIN:      16100
          - NAVER:      27500      - NAVER:      25500
                   2 RECORDS                  2 RECORDS
```

COMPUTE、MOVE、および COMPRESS ステートメント

次の例は、「*データ計算*」セクションで参照されています。

WRITEX11 - WRITE ステートメント（*n***X**、*n*/*n*、および **COMPRESS** を含む）

```
** Example 'WRITEX11': WRITE (with nX, n/n and COMPRESS)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 SALARY       (1)
  2 FIRST-NAME
  2 NAME
  2 CITY
  2 ZIP
  2 CURR-CODE    (1)
  2 JOB-TITLE
  2 LEAVE-DUE
  2 ADDRESS-LINE (1)
*
1 #SALARY        (A8)
1 #FULL-NAME     (A25)
1 #FULL-CITY     (A25)
1 #FULL-SALARY   (A25)
1 #VACATION      (A16)
END-DEFINE
*
READ (3) EMPLOY-VIEW LOGICAL BY PERSONNEL-ID = '2001800'
  MOVE SALARY(1) TO  #SALARY
  COMPRESS FIRST-NAME NAME                    INTO #FULL-NAME
  COMPRESS ZIP        CITY                    INTO #FULL-CITY
  COMPRESS 'SALARY  :'  CURR-CODE(1)  #SALARY INTO #FULL-SALARY
  COMPRESS 'VACATION:'  LEAVE-DUE    'DAYS'   INTO #VACATION
  /*
  DISPLAY NOTITLE 'NAME AND ADDRESS' NAME
               5X 'PERS-NO.'         PERSONNEL-ID
               3X 'JOB TITLE'        JOB-TITLE (LC='JOB     : ')
  WRITE   1/5 #FULL-NAME        1/37  #FULL-SALARY
          2/5 ADDRESS-LINE(1)   2/37  #VACATION
          3/5 #FULL-CITY
  SKIP 1
```

```
END-READ
END
```

プログラム WRITEX11 の出力：

```
  NAME AND ADDRESS       PERS-NO.               JOB TITLE
--------------------     --------   -----------------------------------

FARRIS                   20018000   JOB     : PROGRAMMER
    JACKIE FARRIS                   SALARY  : USD 30500
    918 ELM STREET                  VACATION: 10 DAY
    32306 TALLAHASSEE

EVANS                    20018100   JOB     : PROGRAMMER
    JO EVANS                        SALARY  : USD 31000
    1058 REDSTONE LANE              VACATION: 11 DAY
    68508 LINCOLN

HERZOG                   20018200   JOB     : PROGRAMMER
    JOHN HERZOG                     SALARY  : USD 31500
    255 ZANG STREET #253            VACATION: 12 DAY
    27514 CHAPEL HILL
```

IFX03 - IF ステートメント

```
** Example 'IFX03': IF
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 BONUS  (1,1)
  2 SALARY (1)
*
1 #INCOME  (N9)
1 #TEXT    (A26)
END-DEFINE
*
WRITE TITLE '-- DISTRIBUTION OF CATALOGS I AND II --' /
*
READ (3) EMPLOY-VIEW BY CITY = 'SAN FRANSISCO'
  COMPUTE #INCOME = BONUS(1,1) + SALARY(1)
  /*
  IF #INCOME > 40000
    MOVE 'CATALOGS I AND II' TO #TEXT
  ELSE
    MOVE 'CATALOG I'         TO #TEXT
  END-IF
  /*
```

```
  DISPLAY NAME 5X 'SALARY' SALARY(1) / BONUS(1,1)
  WRITE T*SALARY '-'(10) /
        16X 'INCOME:' T*SALARY #INCOME 3X #TEXT /
        16X '='(19)
  SKIP 1
END-READ
END
```

プログラム IFX03 の出力：

```
                    -- DISTRIBUTION OF CATALOGS I AND II --

        NAME               SALARY
                           BONUS
--------------------     ----------

COLVILLE JR                   56000
                                  0
                           ----------
                 INCOME:       56000   CATALOGS I AND II
                 ==================

RICHMOND                       9150
                                  0
                           ----------
                 INCOME:        9150   CATALOG I
                 ==================

MONKTON                       13500
                                600
                           ----------
                 INCOME:       14100   CATALOG I
                 ==================
```

COMPRX03 - COMPRESS ステートメント（パラメータ **LC** および **TC** を使用）

```
** Example 'COMPRX03': COMPRESS (using parameters LC and TC)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 SALARY     (1)
  2 CURR-CODE  (1)
  2 LEAVE-DUE
  2 NAME
  2 FIRST-NAME
  2 JOB-TITLE
*
1 #SALARY      (N9)
```

```
1 #FULL-SALARY (A25)
1 #VACATION    (A11)
END-DEFINE
*
READ (3) EMPLOY-VIEW WITH CITY = 'BOSTON'
  MOVE SALARY(1) TO #SALARY
  COMPRESS 'SALARY  :' CURR-CODE(1) #SALARY INTO #FULL-SALARY
  COMPRESS 'VACATION:' LEAVE-DUE            INTO #VACATION
  /*
  DISPLAY NOTITLE NAME FIRST-NAME
           'JOB DESCRIPTION' JOB-TITLE  (LC='JOB     : ')  /
           '/'               #FULL-SALARY                   /
           '/'               #VACATION  (TC='DAYS')
  SKIP 1
END-READ
END
```

プログラム COMPRX03 の出力：

```
        NAME             FIRST-NAME               JOB DESCRIPTION
-------------------- -------------------- ----------------------------------

SHAW                 LESLIE               JOB     : SECRETARY
                                          SALARY  : USD 18000
                                          VACATION: 2DAYS

STANWOOD             VERNON               JOB     : PROGRAMMER
                                          SALARY  : USD 31000
                                          VACATION: 1DAYS

CREMER               WALT                 JOB     : SECRETARY
                                          SALARY  : USD 20000
                                          VACATION: 3DAYS
```

システム変数

次の例は、「*システム変数とシステム関数*」で参照されているものです。

EDITMX05 - 編集マスク（日付および時刻システム変数の **EM**）

```
** Example 'EDITMX05': Edit mask (EM for date and time system variables)
************************************************************************
WRITE NOTITLE //
  'DATE INTERNAL :' *DATX (DF=L) /
  '              :' *DATX (EM=N(9)' 'ZW.'WEEK 'YYYY)  /
  '              :' *DATX (EM=ZZJ'.DAY 'YYYY)        /
  '     ROMAN    :' *DATX (EM=R)  /
  '     AMERICAN :' *DATX (EM=MM/DD/YYYY)      12X 'OR  ' *DAT4U /
  '     JULIAN   :' *DATX (EM=YYYYJJJ)         15X 'OR  ' *DAT4J /
  '     GREGORIAN:' *DATX (EM=ZD.''L(10)''YYYY) 5X 'OR  ' *DATG ///
  'TIME INTERNAL :' *TIMX                      14X 'OR  ' *TIME /
  '              :' *TIMX (EM=HH.II.SS.T) /
  '              :' *TIMX (EM=HH.II.SS' 'AP) /
  '              :' *TIMX (EM=HH)
END
```

プログラム EDITMX05 の出力：

```
DATE INTERNAL : 2004-12-13
              : Monday 51.WEEK 2004
              : 348.DAY 2004
     ROMAN    : MMIV
     AMERICAN : 12/13/2004           OR   12/13/2004
     JULIAN   : 2004348              OR   2004348
     GREGORIAN: 13.December2004      OR   13December 2004


TIME INTERNAL : 14:36:58             OR   14:36:58.8
              : 14.36.58.8
              : 02.36.58 PM
              : 14
```

READX04 - READ ステートメント（**FIND** およびシステム変数 ***NUMBER** と ***COUNTER** との組み合わせ）

```
** Example 'READX04': READ (in combination with FIND and the system
**                     variables *NUMBER and *COUNTER)
************************************************************************
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 10
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  FD. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      ENTER
    END-NOREC
    /*
    DISPLAY NOTITLE
            *COUNTER (RD.)(NL=8) NAME         (AL=15) FIRST-NAME (AL=10)
            *NUMBER  (FD.)(NL=8) *COUNTER (FD.)(NL=8) MAKE
  END-FIND
END-READ
END
```

プログラム READX04 の出力：

```
  CNT          NAME       FIRST-NAME   NMBR      CNT             MAKE
--------- --------------- ---------- --------- --------- --------------------

        1 JONES           VIRGINIA           1         1 CHRYSLER
        2 JONES           MARSHA             2         1 CHRYSLER
        2 JONES           MARSHA             2         2 CHRYSLER
        3 JONES           ROBERT             1         1 GENERAL MOTORS
        4 JONES           LILLY              2         1 FORD
        4 JONES           LILLY              2         2 MG
        5 JONES           EDWARD             1         1 GENERAL MOTORS
        6 JONES           MARTHA             1         1 GENERAL MOTORS
        7 JONES           LAUREL             1         1 GENERAL MOTORS
        8 JONES           KEVIN              1         1 DATSUN
        9 JONES           GREGORY            1         1 FORD
       10 JOPER           MANFRED            0         0
```

WTITLX01 - WRITE TITLE ステートメント（***PAGE-NUMBER** を含む）

```
** Example 'WTITLX01': WRITE TITLE (with *PAGE-NUMBER)
************************************************************************
DEFINE DATA LOCAL
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
  2 YEAR
  2 MAINT-COST (1)
END-DEFINE
*
LIMIT 5
*
```

```
READ VEHIC-VIEW
END-ALL
SORT BY YEAR USING MAKE MAINT-COST (1)
  DISPLAY NOTITLE YEAR MAKE MAINT-COST (1)
  AT BREAK OF YEAR
    MOVE 1 TO *PAGE-NUMBER
    NEWPAGE
  END-BREAK
  /*
  WRITE TITLE LEFT JUSTIFIED
        'YEAR:' YEAR 15X 'PAGE' *PAGE-NUMBER
END-SORT
END
```

プログラム WTITLX01 の出力：

```
YEAR:  1980               PAGE      1
YEAR          MAKE          MAINT-COST
----- -------------------- ----------

 1980 RENAULT                   20000
 1980 RENAULT                   20000
 1980 PEUGEOT                   20000
```

システム関数

次の例は、「*システム変数とシステム関数*」で参照されているものです。

ATBREX06 - AT BREAK OF ステートメント（**NMIN**、**NAVER**、**NCOUNT** を **MIN**、**AVER**、**COUNT** と比較）

```
** Example 'ATBREX06': AT BREAK OF (comparing NMIN, NAVER, NCOUNT with
**                       MIN, AVER, COUNT)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 SALARY (1:2)
END-DEFINE
*
WRITE TITLE '-- SALARY STATISTICS BY CITY --' /
*
READ (2) EMPLOY-VIEW WITH CITY = 'NEW YORK'
  DISPLAY CITY  'SALARY (1)' SALARY(1) 15X 'SALARY (2)' SALARY(2)
  AT BREAK OF CITY
    WRITE /
```

```
      14T 'S A L A R Y   (1)'         39T 'S A L A R Y  (2)'          /
      13T '-   MIN:' MIN(SALARY(1))   38T '-   MIN:' MIN(SALARY(2))   /
      13T '-  AVER:' AVER(SALARY(1))  38T '-  AVER:' AVER(SALARY(2))  /
      16T COUNT(SALARY(1)) 'RECORDS'  41T COUNT(SALARY(2)) 'RECORDS'  //
      13T '-  NMIN:' NMIN(SALARY(1))  38T '-  NMIN:' NMIN(SALARY(2))  /
      13T '- NAVER:' NAVER(SALARY(1)) 38T '- NAVER:' NAVER(SALARY(2)) /
      16T NCOUNT(SALARY(1)) 'RECORDS' 41T NCOUNT(SALARY(2)) 'RECORDS'
  END-BREAK
END-READ
END
```

プログラム ATBREX06 の出力：

```
                      -- SALARY STATISTICS BY CITY --

        CITY         SALARY (1)               SALARY (2)
-------------------- ----------               ----------

NEW YORK                  17000                    16100
NEW YORK                  38000                    34900

             S A L A R Y   (1)        S A L A R Y  (2)
            -   MIN:      17000      -   MIN:      16100
            -  AVER:      27500      -  AVER:      25500
                     2 RECORDS                2 RECORDS

            -  NMIN:      17000      -  NMIN:      16100
            - NAVER:      27500      - NAVER:      25500
                     2 RECORDS                2 RECORDS
```

ATENPX01 - AT END OF PAGE ステートメント（**DISPLAY** の **GIVE SYSTEM FUNCTIONS** 経由で使用可能なシステム関数を含む）

```
** Example 'ATENPX01': AT END OF PAGE (with system function available
**                      via GIVE SYSTEM FUNCTIONS in DISPLAY)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY (1)
END-DEFINE
*
READ (10) EMPLOY-VIEW BY PERSONNEL-ID = '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
          NAME JOB-TITLE 'SALARY' SALARY(1)
  /*
  AT END OF PAGE
```

```
    WRITE / 24T 'AVERAGE SALARY: ...' AVER(SALARY(1))
  END-ENDPAGE
END-READ
END
```

プログラム ATENPX01 の出力：

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

CREMER               ANALYST                        34000
MARKUSH              TRAINEE                        22000
GEE                  MANAGER                        39500
KUNEY                DBA                            40200
NEEDHAM              PROGRAMMER                     32500
JACKSON              PROGRAMMER                     33000
PIETSCH              SECRETARY                      22000
PAUL                 SECRETARY                      23000
HERZOG               MANAGER                        48500
DEKKER               DBA                            48000

                       AVERAGE SALARY: ...      34270
```

索引