Natural for Windows

ファーストステップ

バージョン 6.3.3

October 2008

Natural

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

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

目次

1 ファーストステップ

このチュートリアルでは、Natural スタジオ（Natural の開発環境）および Natural を使用したプログラミングについて簡単に概要を説明します。

Important: 次のトピックを以下に示す順に読み、トピックに含まれているすべての演習をこのチュートリアルで示す順に行うことが重要です。 演習をスキップすると、問題が生じる可能性があります。

	このチュートリアルについて	前提条件およびこのチュートリアルで習得する内容。
	Natural スタジオの基本	Natural スタジオの起動方法。 ライブラリワークスペースに関する情報およびコマンドを発行するための異なる方法。 このチュートリアルで使用するライブラリを作成する方法。 Natural のプログラミングモードおよびこのチュートリアルで必要なモードに関する情報。
	Hello World!	最初の簡単なプログラムを作成、実行、および格納する方法。 現在のライブラリの内容を表示する方法。 ワークスペースを制御する一部のオプションに関する情報。
	データベースへのアクセス	デモデータベースに関する情報。 特定のデータをデータベースから読み込み出力を表示する方法。
	ユーザー入力	情報の入力を求めるプロンプトをユーザーに対して表示する方法およびユーザー入力のマップを設計する方法。 ユーザーが指定しない場合でも特定の値（ここでは終了名）を常に確実に使用する方法。
	ループおよびラベル	繰り返しループおよび異なるループのラベルを定義する方法。 特定の情報（ここではユーザーが入力する開始名）が見つからない場合にメッセージを表示する方法。
	インラインサブルーチン	インラインサブルーチン（プログラムで直接コーディングするサブルーチン）を定義および起動する方法。
	処理ルールとヘルプルーチン	処理ルール（ここではユーザーが開始名を指定しない場合に表示するメッセージ）およびヘルプルーチン（ここではユーザーが開始名を入力する必要があるフィールドのヘルプテキスト）を定義する方法。

	ローカルデータエリア	フィールド定義をプログラムからプログラム外のローカルデータエリアに再配置する方法。
	グローバルデータエリア	複数のプログラムまたはルーチンで共有できるグローバルデータエリアを定義する方法。
	外部サブルーチン	外部サブルーチン（プログラム外に別のオブジェクトとして格納されるサブルーチン）を定義および起動する方法。
	サブプログラム	サブプログラムのパラメータデータエリアを定義する方法。 サブプログラムを定義および起動する方法。

2 このチュートリアルについて

初めてのユーザーの場合は、このチュートリアル全体を実行して、Naturalプログラミング環境の特定の機能についての基本的な理解を習得することをお勧めします。

Microsoft Windows について基本的に理解していることが前提となります。

チュートリアルで提供されている例の画面のレイアウトと、ここで説明しているNaturalの動作は、ユーザーの結果と異なる場合があります。 例えば、コマンドまたはメッセージ行が異なる画面の位置に表示されたり、Naturalコマンドの実行がセキュリティコントロールによって保護されている場合があります。 環境のデフォルト設定は、Natural管理者が設定したシステムパラメータによって異なります。

このchapterでは、次のトピックについて説明します。

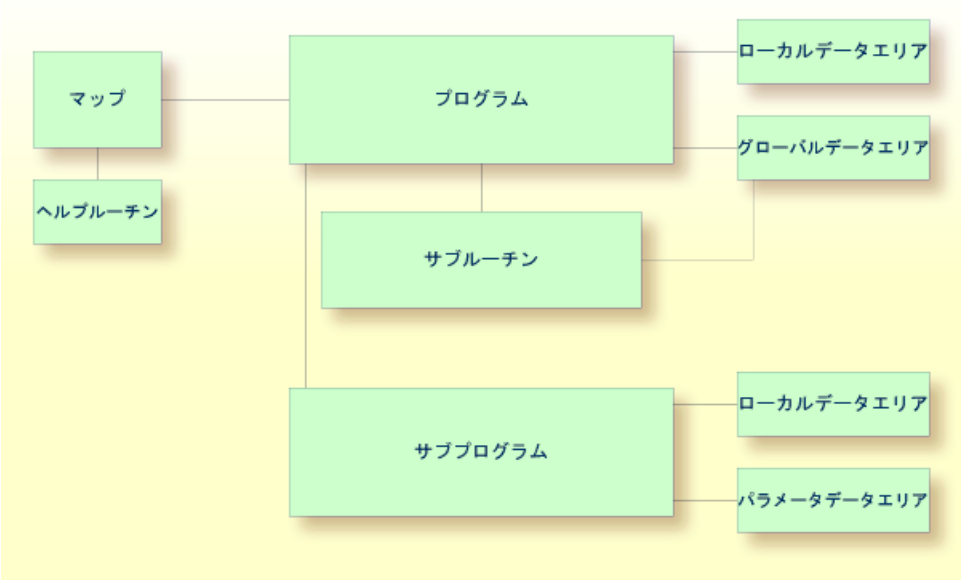
前提条件

このチュートリアルのすべてのステップを実行するには、デモデータベース `SAG-DEMO-DB` を（Windows または UNIX ではローカルに）インストールし、それがアクティブになっている必要があります。 このデータベースは Adabas ではインストールされますが、Natural では自動的にインストールされません。

サンプルのアプリケーションについて

このチュートリアルでは、モジュールのグループとしてアプリケーションをどのように構成できるかを示します。 アプリケーションの構築方法の例を示すわけではありません。

最初の簡単なHello Worldプログラムを作成した後、データベースから従業員情報を読み込み出力を表示するプログラムを作成します。 出力用の開始名および終了名の入力を指示するプロンプトをユーザーに対して表示します。 プログラムの特定の部分を外部モジュールに移動することにより、プログラムをステップごとに強化していきます。 このチュートリアルの演習をすべて完了すると、アプリケーションは次のような構成になります。



Note: このチュートリアルでは、一般に文字型環境（メインフレームなど）で使用されるマップの作成方法について説明します。 グラフィカルユーザーインターフェイスの場合は、ダイアログを作成します。 ただし、これはこのチュートリアルの範囲外です。

Tip: すべてのプログラムコードを自分で入力したくない場合は、チュートリアルから Natural エディタにコード例をコピーできます（コピーおよび貼り付けを行う Ctrl + C および Ctrl + V などの Windows 標準のキーの組み合せを使用）。

最初の演習「*Natural スタジオの基本*」に進みます。

3 Natural スタジオの基本

このchapterでは、次のトピックについて説明します。

Natural スタジオの起動

Natural をインストールすると、対応するフォルダが［スタート］メニューの［プログラム］フォルダに自動的に表示されます。 このフォルダには、Natural の開発環境である Natural スタジオを含む、Natural のショートカットが含まれています。 インストール時に指定すれば、Windows のデスクトップで複数のショートカットを使用することもできます。

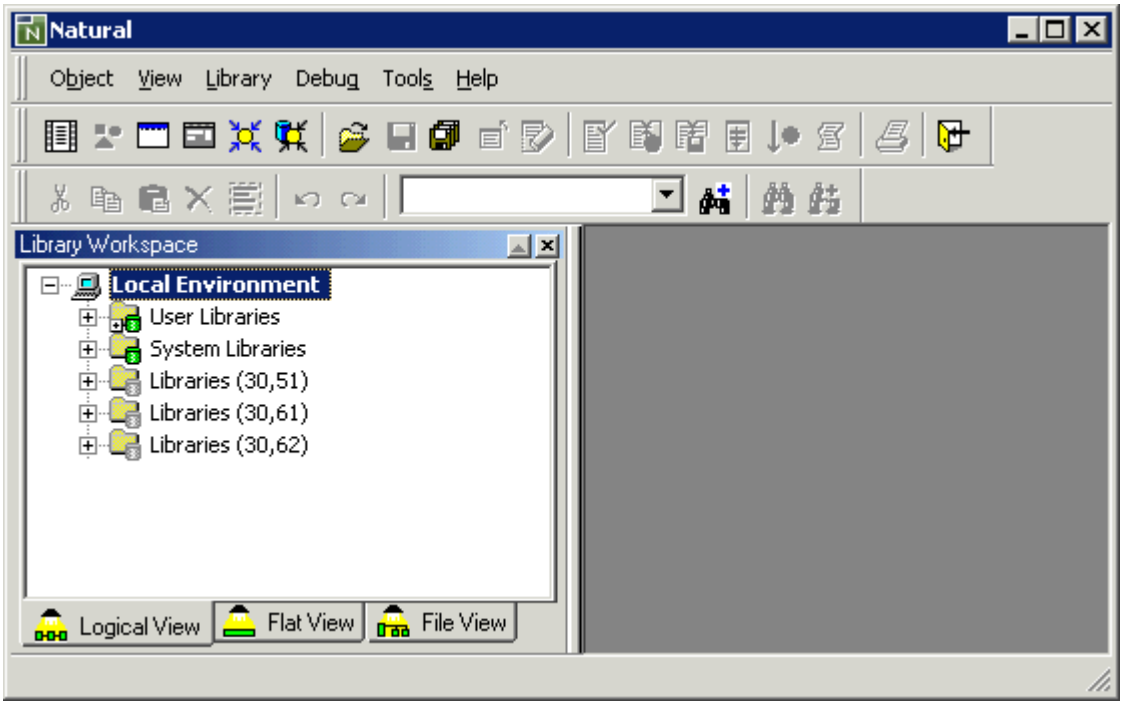
手順 3.1. Natural スタジオを起動するには

■ ［スタート］メニューから、［プログラム］>［**Software AG Natural** *n.n*］>［**Natural**］を選択します。

Or:

Windows のデスクトップで次のショートカットを使用します（インストール時に指定した場合にのみ使用可能）。

Natural スタジオウィンドウが表示されます。



Naturalスタジオを初めて起動した場合は、ライブラリワークスペースを含むローカル環境のみがウィンドウに表示されます。

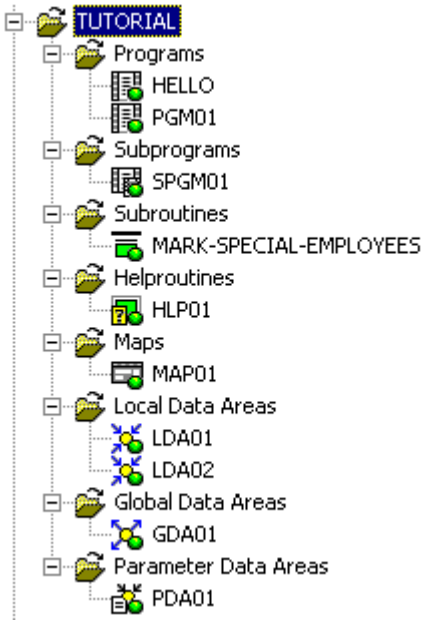
ライブラリワークスペース

アプリケーションの作成に必要な Natural オブジェクトはすべて、Natural システムファイルの Natural ライブラリに格納されます。 システムファイルには、システムプログラム用のもの（FNAT）とユーザー作成プログラム用のもの（FUSER）があります。

したがって、Naturalではシステムライブラリとユーザーライブラリが区別されます。 システムライブラリは "SYS" という文字で始まり、Software AG 専用に確保されています。 ユーザーライブラリには、アプリケーションを構成するユーザー定義オブジェクト（プログラムおよびマップなど）がすべて含まれます。 ユーザーライブラリの名前を "SYS" という文字で始めることはできません。

ライブラリワークスペースでは異なるビューを使用できます。このチュートリアルの演習では、論理ビューで作業を行う必要があります。 論理ビューには、ライブラリ用に異なるノードが表示されます。 ライブラリ内のオブジェクトは、タイプ別に異なるフォルダにグループ化されます。

このチュートリアルの演習をすべて完了すると、ユーザーライブラリ **TUTORIAL** のノードに次のフォルダおよびオブジェクトが含まれます。



コマンドの発行

他の Windows アプリケーションと同様に、Natural スタジオのほとんどのコマンドはいくつかの異なる方法で発行できます。具体的には、メニューバーまたはコンテキストメニューからの選択、ツールバーボタンの選択、またはキーの組み合わせによってコマンドを発行できます。 ただし、このチュートリアルでは、同じコマンドの発行に対してすべての方法を示しません。 一般的に使用される方法（ほとんどの場合はコンテキストメニューおよびツールバーボタン）のみを示します。

コンテキストメニュー（ライブラリワークスペース内のオブジェクト用など）を表示するには、オブジェクトを選択した後に、右マウスボタンをクリックするか、または Shift キーを押したまま F10 キーを押します。

いくつかのメニューまたはツールバーは、特定のコンテキストでのみ表示されます。 例えば、［プログラム］メニューは、プログラムエディタを操作しておりプログラムエディタウィンドウがアクティブになっている場合にのみメニューバーに表示されます。

ユーザーライブラリの作成

TUTORIAL という名前のユーザーライブラリを作成します。 このライブラリに、このチュートリアルで作成するすべての Natural オブジェクトを格納します。

手順 3.2. ユーザーライブラリを作成するには

1 論理ビューで、トップノード **Local Environment** の真下にある **User Libraries** という名前のノードを選択します。

2 コンテキストメニューから、［新規作成］を選択します。

デフォルト名 **USRNEW** を持つ新しいライブラリがツリーに表示されます。

3 ライブラリ名として「TUTORIAL」と指定し、Enter キーを押します。

プログラミングモード

Natural には、2 つの異なるプログラミングモードがあります。

- 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は単独で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。

Important: このチュートリアルでは、ストラクチャードモードがアクティブになっている必要があります。 プログラムをレポーティングモードで実行しようとすると、END-IF、END-READ、および END-REPEAT でエラーになります。

手順 3.3. ストラクチャードモードがアクティブであるかどうかを確認するには

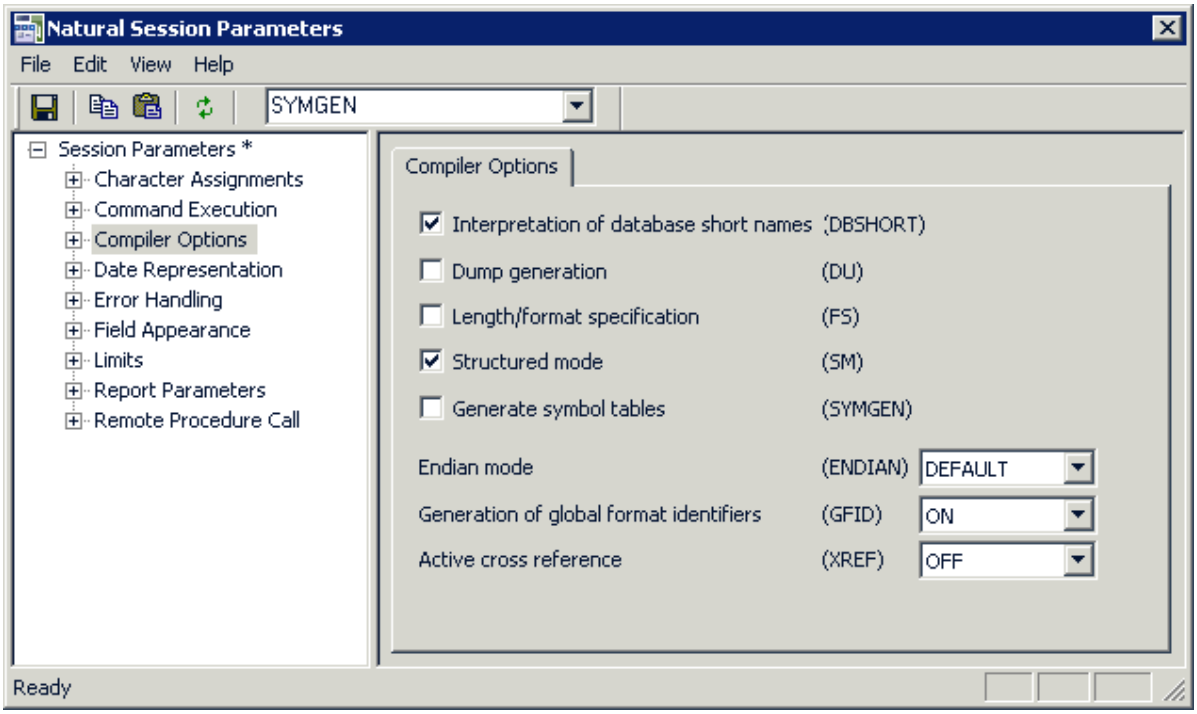
1 ［ツール］メニューの［セッションパラメータ］を選択します。

［**Natural** セッションパラメータ］ダイアログボックスが表示されます。

これらのパラメータは、Natural のインストール時に Natural 管理者によって Natural の全ユーザーに有効なデフォルト値に設定されます。 これらに加えた変更は、現在のセッションでのみ有効です。

2 左側のツリーで［コンパイラオプション］を選択します。

コンパイラオプションが右側に表示されます。



［ストラクチャードモード］チェックボックスがすでにオンになっている場合、以下の手順は不要となり、チェックボックスを閉じることができます。 オンになっていない場合は、以下の手順に従ってください。

3 ［ストラクチャードモード］チェックボックスをオンにします。

4 ［ファイル］メニューの［保存］を選択します。

5 ダイアログボックスを閉じます。

最初のプログラム「*Hello World!*」に進みます。

4 Hello World!

このchapterには次の演習が含まれています。

プログラムの作成

"Hello World!" と表示する最初の簡単なプログラムを作成します。 このプログラムは、前に作成したライブラリに格納されます。

▶手順 4.1. 新しいプログラムを作成するには

1 TUTORIAL という名前のライブラリをライブラリワークスペースで選択します。

2 コンテキストメニューから、［新規作成］>［プログラム］を選択します。

Or:

次のツールバーボタンを選択します。

プログラムエディタが表示されます。 現時点では空になっています。

3 プログラムエディタで次のコードを入力します。

```
* The "Hello world!" example in Natural.
*
DISPLAY "Hello world!"
END /* End of program
```

コメント行はアスタリスク（*）で始まり、少なくとも 1 つの空白または 2 つ目のアスタリスクが後に続きます。 空白または 2 つ目のアスタリスクを入力し忘れると、システム変数を指定したとみなされ、エラーになります。

プログラムに空行を挿入する場合は、コメント行として定義する必要があります。これは、異なるプラットフォーム（Windows、メインフレーム、UNIX、またはOpenVMS）からプログラムにアクセスする場合に便利です。 例えば、メインフレームバージョンの Natural では、デフォルトにより、Enter キーを押すと空行が自動的に削除されます。

また、ステートメント行の最後にコメントを挿入することもできます。 この場合、コメントはスラッシュで始まり、アスタリスクが後に続きます（/*）。

出力に表示するテキストは、DISPLAY ステートメントで定義されます。 表示テキストは引用符で囲みます。

END ステートメントは、Natural プログラムの物理的な終わりをマークするために使用します。 各プログラムは END で終了する必要があります。

プログラムの実行

システムコマンド RUN では、プログラムコードのエラーをチェックするシステムコマンド CHECK が自動的に呼び出されます。 エラーが検出されなければ、その時点でプログラムがコンパイルされ、実行されます。

Notes:

1. システムコマンドは、メインフレームバージョンの Natural でも使用できます。 Windows では、［オブジェクト］メニューから選択することによって、システムコマンドが起動されます。
2. ［オブジェクト］メニューでは、CHECK を別のコマンドとして使用することもできます。
3. Natural には、格納バージョンのプログラムを使用するシステムコマンド EXECUTE も用意されています（プログラムの格納についてはこのチュートリアルで後述）。 これに対し、RUN コマンドでは、最新の修正を含むプログラムが常に使用されます。

手順 4.2. プログラムを実行するには

1 ［オブジェクト］メニューの［**Run**］を選択します。

Or:

次のツールバーボタンを選択します。

コードが構文的に正しければ、定義したテキストが出力に表示されます。

```
Page      1                                                  05-03-11  12:07:25

Hello world!
```

2 Enter キーを押して、プログラムエディタに戻ります。

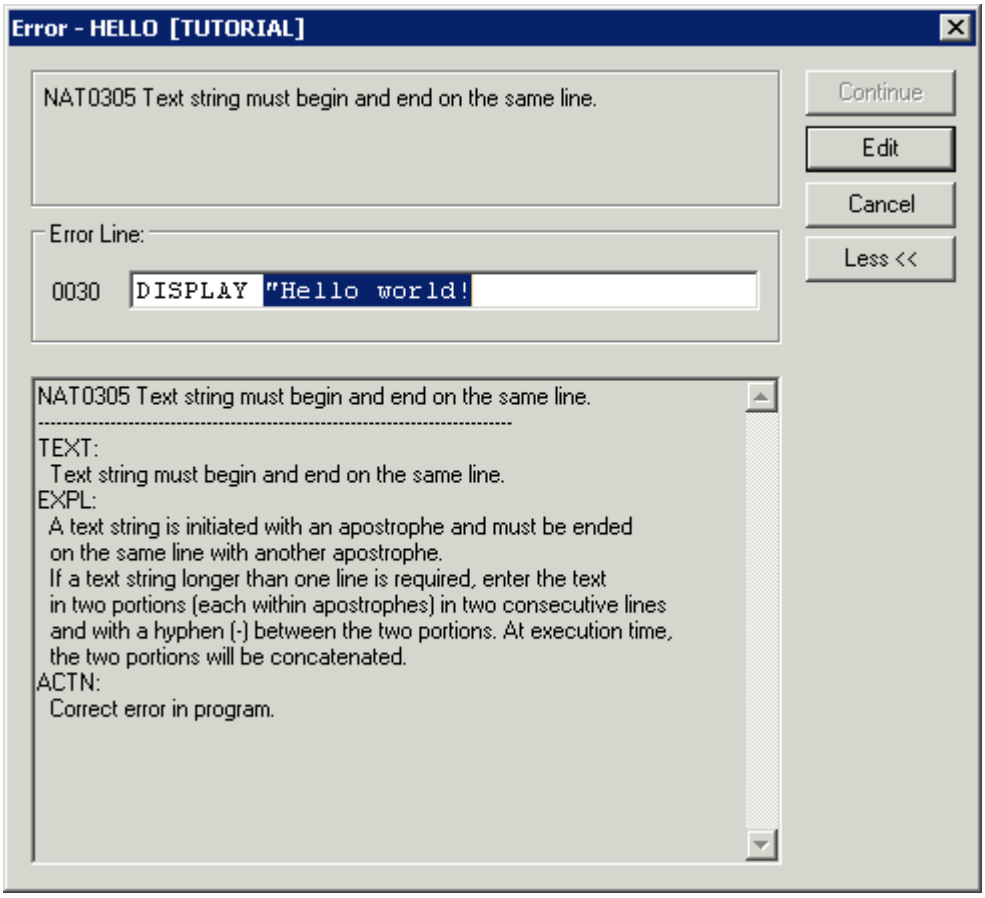
プログラムエラーの修正

Hello World プログラムでエラーを作成してから、プログラムをもう一度実行します。

手順 4.3. エラーを修正するには

1 DISPLAY ステートメントを含む行で 2 つ目の引用符を削除します。
2 上記の手順で、プログラムをもう一度実行します。

エラーが検出されると、エラー情報を示すダイアログボックスが表示されます。



3 ダイアログボックスでエラーを修正します。つまり、欠けている引用符を行の最後に挿入します。

4 ［継続］ボタンを選択して、次のエラーを見つけます。

この場合、他のエラーは検出されず出力が表示されます。

5 Enter キーを押して、プログラムエディタに戻ります。

Note: ［継続］ボタンの代わりに、［編集］ボタンを選択することもできます。ダイアログボックスが閉じるので、プログラムエディタでエラーを直接修正できます。

プログラムの格納

プログラムを格納すると、プログラムがコンパイルされ、ソースコードと生成プログラムの両方が Natural システムファイルに格納されます。

RUN コマンドと同様に、システムコマンド STOW でも CHECK コマンドが自動的に呼び出されます。プログラムは構文的に正しい場合にのみ格納されます。

Note: プログラムに構文エラーが含まれていてもプログラムに加えた変更を保存する（作業を翌日まで停止する場合など）には、［オブジェクト］メニューから起動できるシステムコマンド SAVE を使用できます。

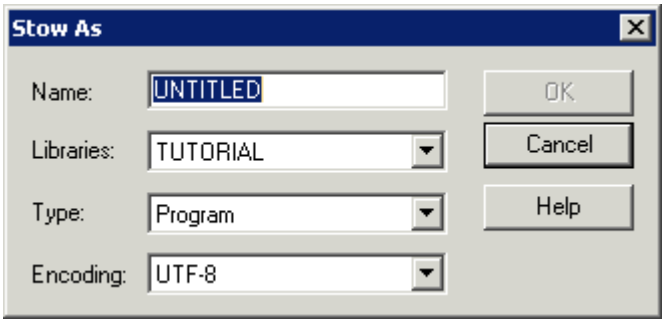
手順 4.4. プログラムを Stow するには

1 ［オブジェクト］メニューの［**Stow**］を選択します。

Or:

次のツールバーボタンを選択します。

プログラムはまだ保存されていないため、［名前を付けて **Stow**］ダイアログボックスが表示されます。



現在選択されているライブラリの名前が、対応するドロップダウンリストボックスに自動的に表示されます。

2 ［名前］テキストボックスに「HELLO」と名前を指定します。

3 ［**OK**］ボタンを選択します。

格納が正常に行われたことを示すメッセージが表示されます。 このメッセージは、特定のワークスペースオプションの設定に応じて、ステータスバーまたはダイアログボックスのどちらかに表示されます（以下を参照）。

ライブラリワークスペースで、**Programs** という名前の新しいノードが **TUTORIAL** ノードの下に表示されます。 このノードには、格納したプログラムが含まれています。

プログラムアイコン上の緑色の点は、オブジェクトに対してソースコードと生成プログラムの両方が存在することを示します。

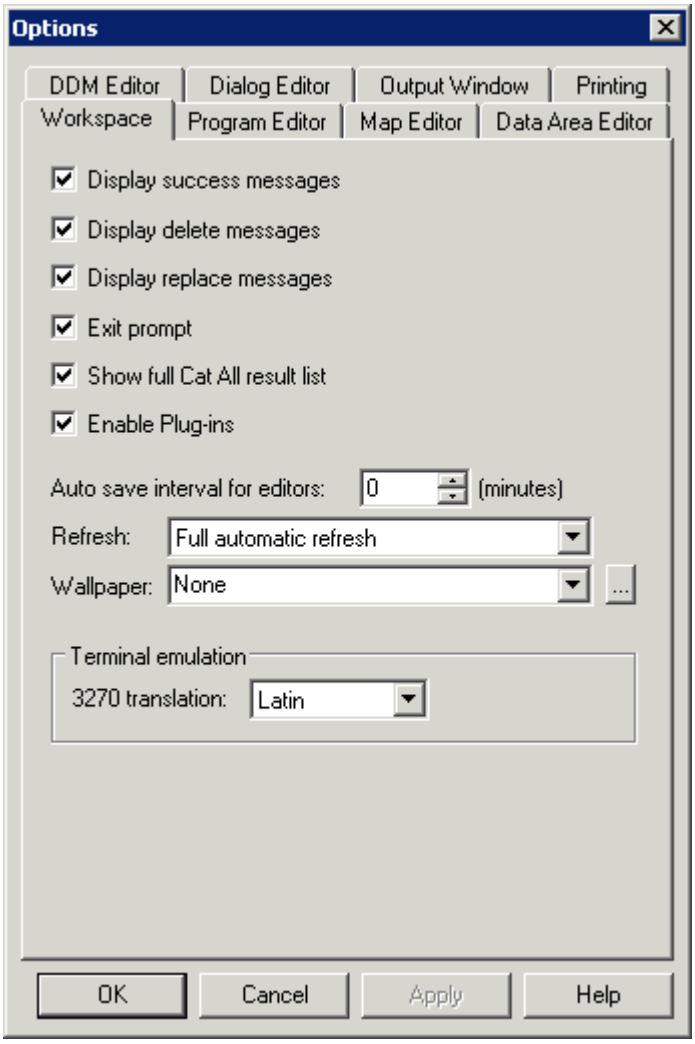
ワークスペースのオプションの設定

ワークスペースオプションの設定をチェックします。

手順 4.5. ワークスペースのオプションをチェックするには

1 ［ツール］メニューの［オプション］を選択します。

2 結果の［オプション］ダイアログボックスに、［ワークスペース］ページが表示されます。



3 ダイアログボックスに完了メッセージを表示する場合は、対応するチェックボックスがオンになっていることを確認します。

Note: プログラムエディタで行番号を表示するかどうかは、［プログラムエディタ］ページのオプションによって制御されます。

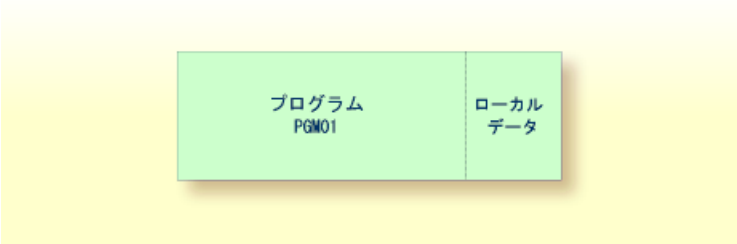
4 ［**OK**］ボタンを選択して変更内容を保存し、ダイアログボックスを閉じます。

次の演習「データベースへのアクセス」に進みます。

5 データベースへのアクセス

特定のデータをデータベースファイルから読み込み、対応する出力を表示する簡単なプログラムを作成します。

以下の演習を完了すると、サンプルのアプリケーションは1つのモジュールでのみ構成されます（プログラムで使用されるデータフィールドはプログラム内で定義されます）。



このchapterには次の演習が含まれています。

デモデータベースの開始

デモデータベース SAG-DEMO-DB は自動的に開始されません。 このチュートリアルの演習に進む前に、デモデータベースが開始していることを確認する必要があります。 デモデータベースが開始していないと、例が動作しません。

次の説明は、Adabas を Windows でローカルにインストールした場合に適用されます。 UNIX バージョンを使用し、デモデータベースが稼動していない場合は、開始するよう UNIX 管理者に依頼してください。

手順 5.1. デモデータベースを開始するには

1 ［スタート］メニューから、［プログラム］>［**Adabas** *n.n*］>［**DBA Workbench**］を選択します。

デモデータベースのステータスは、結果のデータベースリストに表示されます。 ステータスが "Active" の場合、以下の手順は不要となり、［DBA Workbench］アプリケーションウィンドウを閉じることができます。

ステータスが "Active" でない場合は、以下の手順に従ってください。

2 データベースリストの［**SAG-DEMO-DB**］を選択します。

3 ［**Database**］メニューの［**Start**］を選択します。

データベースが開始したことを示すダイアログボックスが表示されます。

4 ［**OK**］ボタンを選択して、ダイアログボックスを閉じます。

5 ［DBA Workbench］アプリケーションウィンドウを閉じます。

新しい名前でのプログラムの保存

このチュートリアルの残りの部分で使用する新しいプログラムを作成します。 新しいプログラムは、Hello World プログラムを新しい名前で保存することによって作成します。

手順 5.2. プログラムを新しい名前で保存するには

1 ［オブジェクト］メニューの［名前を付けて保存］を選択します。

Tip: プログラムエディタが選択されていることを確認します。 プログラムエディタが選択されていないと、上記のコマンドを使用できません。

［名前をつけて保存］ダイアログボックスが表示されます。

2 プログラムの新しい名前として「PGM01」と指定します。

3 ［**OK**］ボタンを選択します。

新しい名前がプログラムエディタのタイトルバーに表示されます。

ライブラリワークスペースで、新しいプログラムが **Programs** ノードに表示されます。 このプログラムはまだ格納されていないため、プログラムアイコンに緑色の点は表示されません。

4 プログラムエディタですべてのコードを削除します（例えば、Ctrlキーを押したままAキーを押してすべてのテキストを選択してから、Delete キーを押します。これは Windows の標準機能です）。

ビューを使用した必須データの定義

プログラムで使用するデータベースファイルおよびフィールドは、プログラムの先頭にある `DEFINE DATA` と `END-DEFINE` との間で指定する必要があります。

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。 DDM には、ファイルの個々のフィールドに関する情報が含まれます。 DDM は通常、Natural 管理者が定義します。

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 DDM からフィールドを指定する必要があります。 システムライブラリ SYSEXDDM には、サンプル DDM が用意されています。 このチュートリアルでは、EMPLOYEES データベースファイル用の DDM を使用します。

必要なフォーマットおよび長さの定義を含め、DDMからプログラムエディタにフィールドをインポートすることができます。

手順 5.3. DEFINE DATA ブロックを指定するには

■ プログラムエディタで次のコードを入力します。

```
DEFINE DATA
LOCAL

END-DEFINE
*
END
```

Tip: Windows バージョンの Natural では、大文字と小文字は区別されません。 しかし、メインフレームバージョンのNaturalでは、キーワードと識別子は常に大文字で入力され、テキスト定数に小文字が含まれることがあります。 したがって、メインフレームでもプログラムを編集する場合には、メインフレームの場合を想定してプログラムコードを入力することをお勧めします。

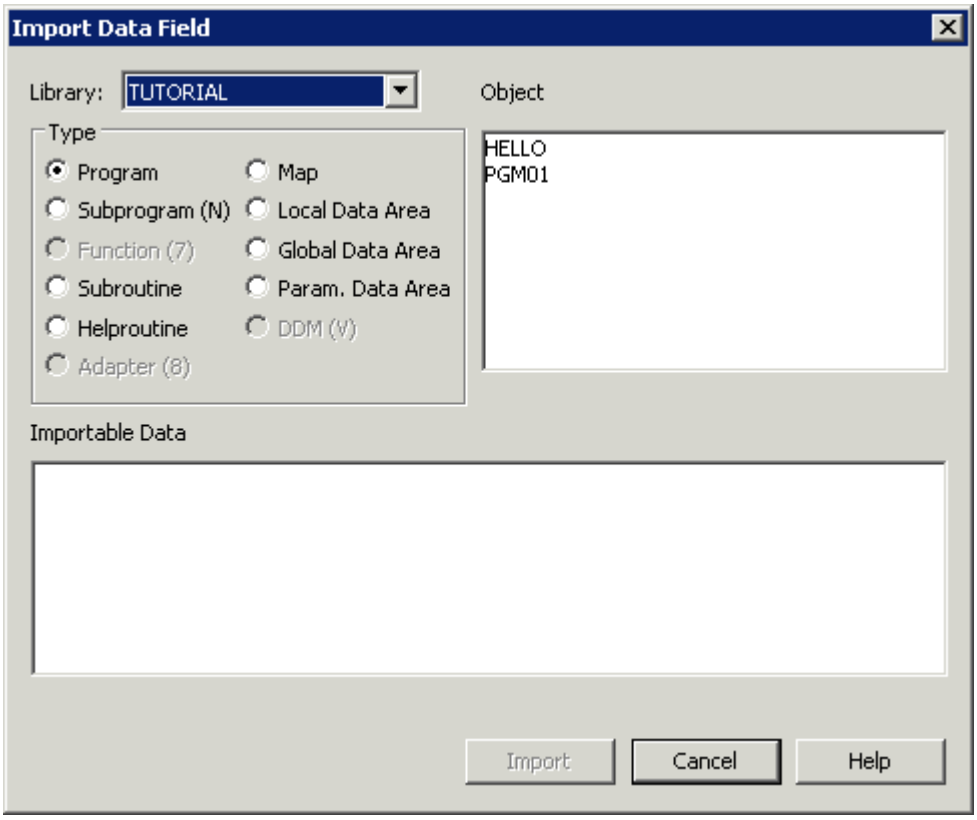
LOCALは、次の手順で定義する変数がこのプログラムにのみ適用されるローカル変数であることを示します。

手順 5.4. DDM からデータフィールドをインポートするには

1 LOCAL の下の行にカーソルを置きます。

2 ［プログラム］メニューの［インポート］を選択します。

［データフィールドのインポート］ダイアログボックスが表示されます。



3 ［ライブラリ］ドロップダウンリストボックスから［**SYSEXDDM**］を選択します。

［**DDM**］オプションボタンが選択されていると、定義済みのすべての DDM が［オブジェクト］リストボックスに表示されます。

4 EMPLOYEES という名前のサンプル DDM を選択します。

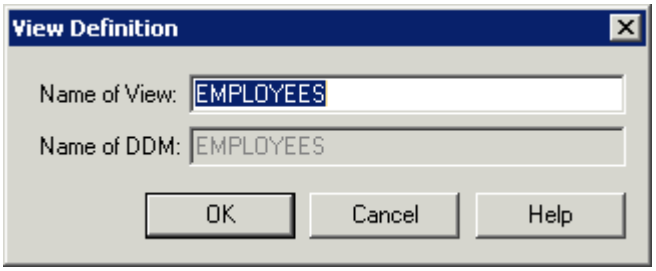
インポートできるデータフィールドがダイアログボックスの下部に表示されます。

5 Ctrl キーを押して、次のフィールドを選択します。

```
FULL-NAME
NAME
DEPT
LEAVE-DATA
LEAVE-DUE
```

6 ［インポート］ボタンを選択します。

［ビューの定義］ダイアログボックスが表示されます。



デフォルトでは、DDM名がビュー名として選択されています。他の任意の名前を指定できます。

7 ビュー名として「EMPLOYEES-VIEW」と入力します。

8 ［**OK**］ボタンを選択します。

［データフィールドのインポート］ダイアログボックスの［キャンセル］ボタンのラベルが［終了］になります。

9 ［終了］ボタンを選択して、［データフィールドのインポート］ダイアログボックスを閉じます。

プログラムエディタで次のコードが挿入されました。

```
1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
  2 FULL-NAME
    3 NAME (A20)
  2 DEPT (A6)
  2 LEAVE-DATA
    3 LEAVE-DUE (N2)
```

最初の行には、ビューの名前およびフィールドが取得されるデータベースファイルの名前が含まれています。 最初の行は、常にレベル 1 で定義されます。 レベルは、行の先頭に示されます。DDM のデータベースフィールドの名前は、レベル 2 および 3 で定義されます。

レベルはフィールドグルーピングと合わせて使用します。2またはそれ以上のレベル番号が付いたフィールドは、それより小さいレベル番号が付いた直前のグループの一部であるとみなされます。 グループで定義すると、グループ名を使用して一連のフィールド（または 1 つのフィールドだけでもよい）を参照することができます。 これは一連の連続フィールドを参照するのに便利で効果的な方法です。

各フィールドのフォーマットおよび長さは、カッコで囲んで示されます。 "A" は英数字を示し、"N" は数字を示します。

データベースからのデータの読み込み

必須データの定義が完了したので、READループを追加します。 このループは、定義されたビューを使用してデータベースファイルからデータを読み込みます。各ループでは、データベースファイルから 1 人の従業員を読み込みます。 この従業員の名前、部署、および休暇の残り日数が表示されます。 すべての従業員が表示されるまで、データが読み込まれます。

Note: トランザクションが中止されたことを示すエラーメッセージが表示されることがあります。これは通常、Adabasによって決められる非アクティビティタイムリミットを超えた場合に発生します。このようなエラーが発生した場合は、最後に実行されたアクションを単に繰り返す（RUN コマンドを再発行するなど）必要があります。

手順 5.5. データベースからデータを読み込むには

1 END-DEFINE の下に次の行を挿入します。

```
READ EMPLOYEES-VIEW BY NAME
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
```

BY NAMEは、データベースから読み込むデータが名前別にアルファベット順にソートされることを示します。

DISPLAY ステートメントは、出力を列形式で配列します。 指定されたフィールドごとに列が作成され、列の上にヘッダーが表示されます。 3X は、列間に 3 文字分の空白が挿入されることを示します。

2 プログラムを実行します。

次の出力が表示されます。

```
Page      1                                                  05-05-18  16:06:49

        NAME           DEPARTMENT   LEAVE
                          CODE       DUE
--------------------   ----------   -----

ABELLAN                PROD04        20
ACHIESON               COMP02        25
ADAM                   VENT59        19
ADKINSON               TECH10        38
ADKINSON               TECH10        18
```

```
ADKINSON                TECH05        17
ADKINSON                MGMT10        28
ADKINSON                TECH10        26
ADKINSON                SALE30        36
ADKINSON                SALE20        37
ADKINSON                SALE20        30
AECKERLE                SALE47        31
AFANASSIEV              MGMT30        26
AFANASSIEV              TECH10        35
AHL                     MARK09        30
AKROYD                  COMP03        20
ALEMAN                  FINA03        20
```

DISPLAY ステートメントの結果、列ヘッダー（DDM から取得）に下線が引かれ、下線とデータの間に 1 行の空行が挿入されます。 各列の幅は、DEFINE DATA ブロックで定義された値（ビューで定義された値）と同じになります。

各ページ上部のタイトルにはページ番号および日時が含まれ、これも DISPLAY ステートメントによって表示されます。

3 Enter キーを繰り返し押して、すべてのページを表示します。

すべての従業員が表示されると、プログラムエディタに戻ります。

Tip: すべての従業員が表示される前にプログラムエディタに戻るには、Esc キーを押します。

データベースからの選択したデータの読み込み

前の出力は非常に長いため、それを制限します。 "Adkinson" で始まり "Bennett" で終わる範囲の名前のデータのみを表示します。 これらの名前はデモデータベースに定義されています。

手順 5.6. 出力をデータの範囲に制限するには

1 新しい変数を使用する前に、それを定義する必要があります。 したがって、LOCAL の下に次の行を挿入します。

```
1 #NAME-START        (A20) INIT <"ADKINSON">
1 #NAME-END          (A20) INIT <"BENNETT">
```

これらはユーザー定義変数であり、デモデータベースには定義されていません。 名前の先頭にあるハッシュ（#）は、ユーザー定義変数とデモデータベースに定義されているフィールドを区別するために使用されていますが、必須文字ではありません。

INIT はフィールドのデフォルト値を定義します。 デフォルト値は、山カッコおよび引用符で囲んで指定する必要があります。

2 READ ステートメントの下に次の行を挿入します。

```
STARTING FROM #NAME-START
ENDING AT #NAME-END
```

プログラムは次のようになります。

```
DEFINE DATA
LOCAL
  1 #NAME-START        (A20) INIT <"ADKINSON">
  1 #NAME-END          (A20) INIT <"BENNETT">
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
END
```

3 プログラムを実行します。

出力が表示されます。 Enter キーを繰り返し押すと、数ページ後（Bennett という名前の最後の従業員のデータが表示された後）でプログラムエディタに戻ります。

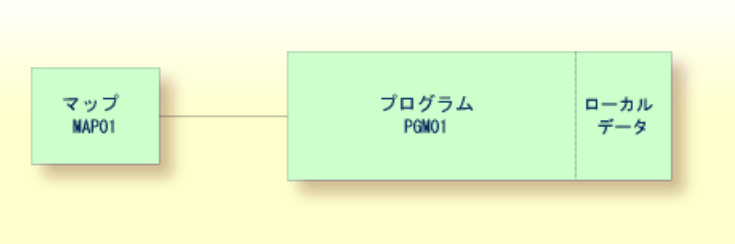
4 プログラムを格納します。

次の演習「ユーザー入力」に進みます。

6 ユーザー入力

データ、つまり出力の開始名と終了名の入力プロンプトをユーザーに対して表示する方法を習得します。

以下の演習を完了すると、サンプルのアプリケーションは次のモジュールで構成されます。



このchapterには次の演習が含まれています。

ユーザー入力の許可

プログラムを修正して、開始名および終了名の入力フィールドが出力に表示されるようにします。 これには、INPUT ステートメントを使用します。

手順 6.1. 入力フィールドを定義するには

1 END-DEFINE の下に次の行を挿入します。

```
INPUT (AD=MT)
  "Start:" #NAME-START /
  "End:   " #NAME-END
```

セッションパラメータ AD は「属性定義」を表し、値"M"は「変更可能出力フィールド」、値 "T" は 「小文字から大文字への変換」 を表します。

AD=MT の値 "M" は、INIT で定義されたデフォルト値（"ADKINSON" および "BENNETT"）が入力フィールドに表示されることを示します。ユーザーは異なる値を入力できます。"M" 値を省略すると、デフォルト値が定義されていても入力フィールドには表示されません。

AD=MT の値 "T" は、小文字の入力がすべて大文字に変換されてから処理されることを示します。 デモデータベースファイル内の名前はすべて大文字で定義されているため、これは重要です。 "T" 値を省略すると、すべての名前を大文字で入力する必要があります。 大文字で入力しないと、指定した名前が検出されなくなります。

"Start:" および "End:" はテキストフィールド（ラベル）であり、 引用符で囲んで指定します。

#NAME-START および #NAME-END はデータフィールド（入力フィールド）であり、任意の開始名および終了名をユーザーを入力できます。

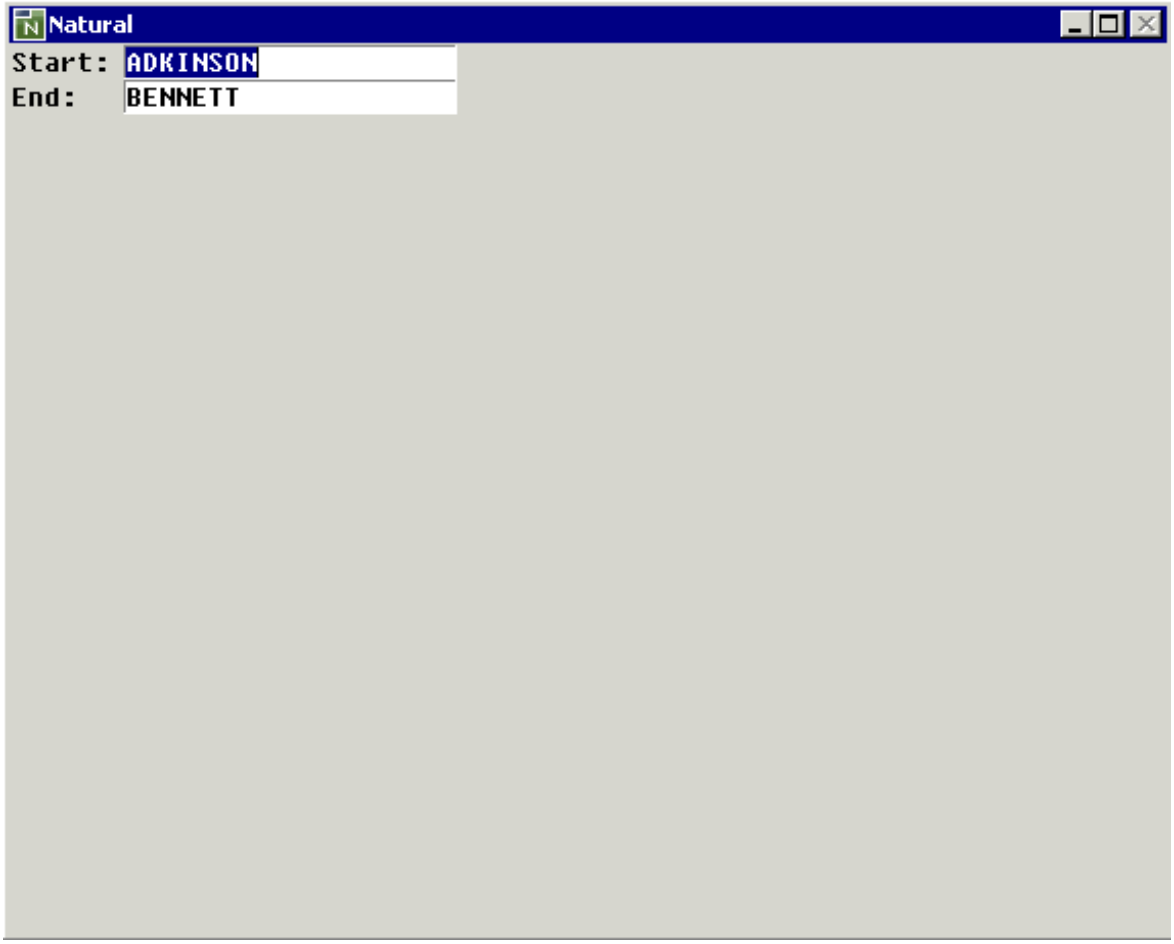
スラッシュ（/）は、後続フィールドを新しい行に表示することを示します。

プログラムは次のようになります。

```
DEFINE DATA
LOCAL
  1 #NAME-START        (A20) INIT <"ADKINSON">
  1 #NAME-END          (A20) INIT <"BENNETT">
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
INPUT (AD=MT)
  "Start:" #NAME-START /
  "End:  " #NAME-END
*
READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
END
```

2 プログラムを実行します。

定義したフィールドが出力に表示されます。



3 デフォルト名を使用し、Enter キーを押します。

従業員のリストが表示されます。

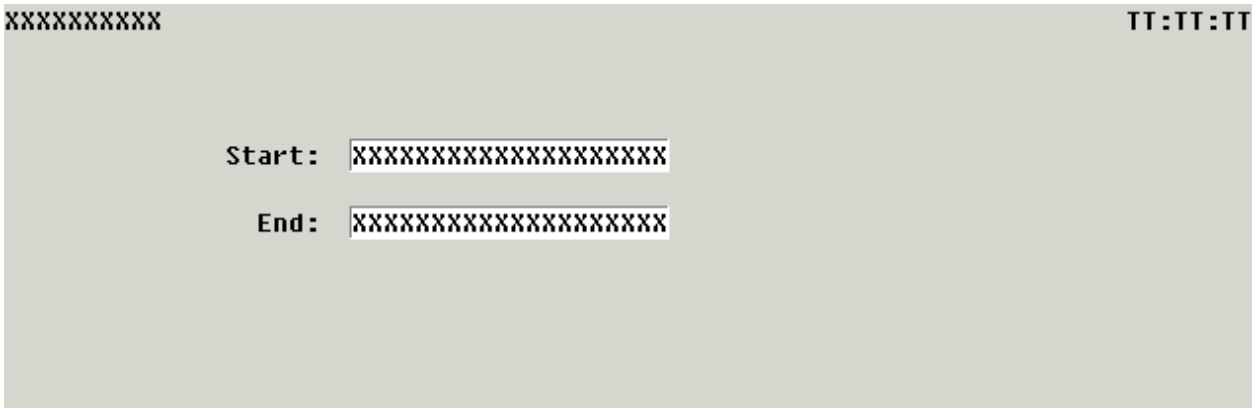
4 プログラムエディタに戻るまで繰り返し Enter キーを押すか、Esc キーを押します。。

5 プログラムを格納します。

ユーザー入力のマップの設計

入力プロンプトをユーザーに表示するさまざまな方法を説明します。 マップエディタを使用して、前にプログラムで定義した同じフィールドを含むマップを作成します。 マップは別のオブジェクトであり、ユーザーインターフェイスのレイアウトをアプリケーションのビジネスロジックから分離するために使用します。

ここで作成するマップは次のようになります。



マップの最初の行には、現在の日時を示すシステム変数が含まれます。 開始名および終了名をユーザーが指定できる 2 つのデータフィールド（入力フィールド）があります。 データフィールドの前にはテキストフィールド（ラベル）があります。

上記のマップには、次の手順が必要です。

- マップの作成
- テキストフィールドの定義
- テキストフィールドのラベルの指定
- データフィールドの定義
- データフィールドの名前および属性の指定
- システム変数の追加
- マップのテスト
- マップの格納

マップの作成

マップエディタを起動し、マップを設計します。

プログラムエディタをバックグラウンドで開いたままにします。

手順 6.2. マップを作成するには

1 ライブラリワークスペースで、プログラムも含まれているライブラリ（**TUTORIAL** ノード）を選択します。

2 コンテキストメニューから、［新規作成］>［マップ］を選択します。

Or:

次のツールバーボタンを選択します。

空のマップエディタウィンドウが表示されます。

テキストフィールドの定義

2 つのテキストフィールド（定数またはラベルとも呼ばれる）をマップに追加します。

手順 6.3. テキストフィールドを定義するには

1 ［挿入］メニューの［テキスト定数］を選択します。

Note: マップエディタがアクティブの場合、メニューバーに異なるメニューが表示されます。（プログラムエディタがアクティブのときに表示される［プログラム］メニューの代わりに）［挿入］、［フィールド］、および［マップ］メニューが表示されます。

Or:

次のツールバーボタンを選択します。

Note: デフォルトでは、このボタンを含むツールバーは、マップエディタウィンドウの右側で縦に表示されます。プログラムエディタがアクティブのときは、異なるツールバーがここに表示されていました。

2 マップエディタウィンドウでテキスト定数を挿入する位置にマウスを移動します。

マウスポインタの表示が十字およびテキスト定数シンボルに変わります。

3 マウスボタンを押し続けます。

4 フィールドが希望の長さになるまでマウスを右にドラッグします。 ここでは、フィールド長は約 10 文字で十分です。

マウスをドラッグすると、現在の長さがアプリケーションウィンドウのステータスバーに表示されます。 フォーマットおよびマップにおける位置もステータスバーに表示されます。テキストフィールドは常にフォーマット A（英数字）になります。

5 マウスボタンを放します。

デフォルトのラベルを持つテキストフィールドが表示されます。 ハンドルは、テキストフィールドが選択されていることを示します。 マウスポインタには引き続き十字およびテキスト定数シンボルが表示され、別のテキストフィールドを直ちに描画できます。

Note: このモードを終了するには、マップエディタ内の他の位置をクリックします。

6 最初のテキストフィールドの下に 2 つ目のテキストフィールドを描画します。

フィールドの開始列に誤りがある場合は、そのフィールドにマウスポインタを置き、マウスボタンを押し続けて目的の位置にドラッグすると、フィールドを移動できます。 マウスボタンを放すと、通常のマウスポインタが再表示されます。

テキストフィールドのラベルの指定

挿入したテキストフィールドには、正しいラベルがまだ付いていません。 ここで、ラベルを指定します。

手順 6.4. ラベルを指定するには

1 最初のテキストフィールドを選択し、コンテキストメニューから［定義］を選択します。

Or:

テキストフィールドをダブルクリックします。

既存のテキストが選択されます。

2 最初のテキストフィールドのラベルに「Start:」と入力し、Enter キーを押します。

テキストフィールド（長さが 10 文字で定義済み）が、この文字列の長さに自動的にサイズ変更されます。

3 上記の手順を繰り返し、2 つ目のテキストフィールドのラベルに「End:」と入力します。

データフィールドの定義

2つのデータフィールドをマップに追加します。これらは、開始名および終了名をユーザーが指定できる入力フィールドです。

データフィールドは 2 つの方法で定義できます。［データフィールド］コマンドでは、データフィールドに正しいフォーマットと長さを定義する必要があります。または、［インポート］>［データフィールド］コマンドでは、データフィールドをリストから選択するだけで、正しいフォーマットと長さが自動的に使用されます。 これらの 2 つの方法について、以下で説明します。

手順 6.5. 長さを指定する必要のあるデータフィールドを定義するには

1 ［挿入］メニューの［データフィールド］を選択します。

Or:

次のツールバーボタンを選択します。

2 開始名用として前に挿入したテキストフィールドの右に、データフィールドを描画します。テキストフィールドと同じように描画します。 データフィールドには 20 文字の長さが必要です（フィールド長が短すぎるまたは長すぎる場合、後述の演習で長さを修正する方法を示します）。 データフィールドは、"X" 文字で自動的に埋められます。

手順 6.6. データフィールドをインポートするには

1 ［挿入］メニューの［インポート］>［データフィールド］を選択します。

［データフィールドのインポート］ダイアログボックスが表示されます。

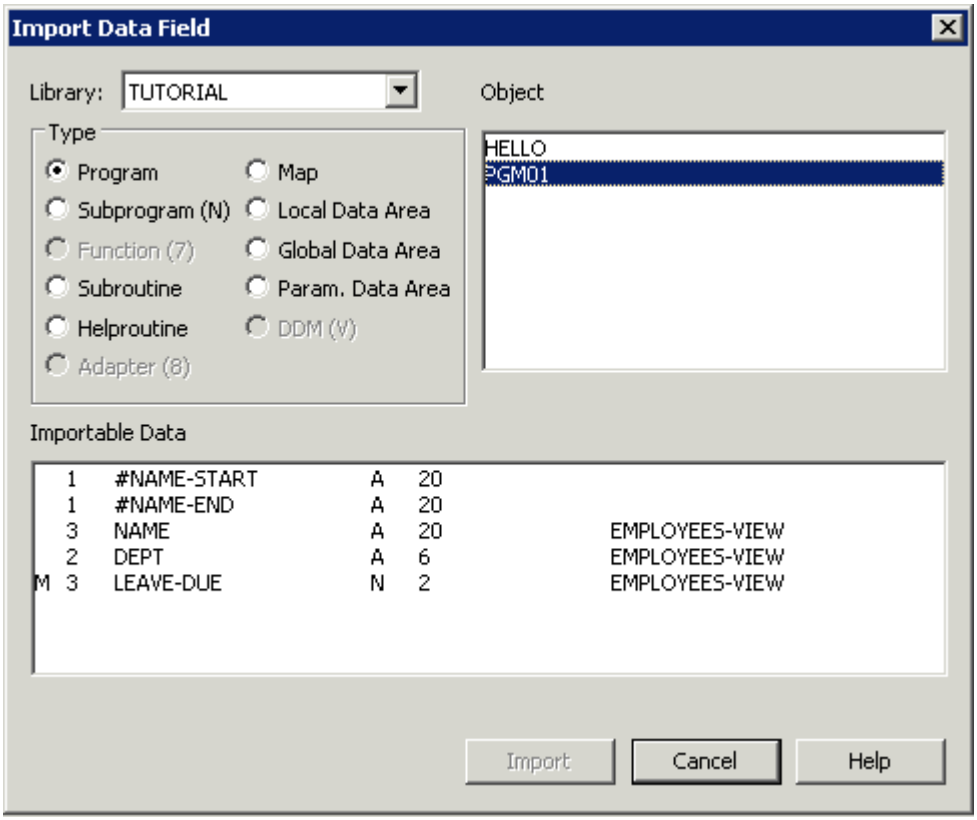
2 ［ライブラリ］ドロップダウンリストボックスから［**TUTORIAL**］を選択します。

3 ［プログラム］オプションボタンを選択します。

ライブラリで現在定義されているすべてのプログラムが、［オブジェクト］リストボックスに表示されます。

4 `PGM01` という名前のプログラムを選択します。

インポートできるデータフィールドがダイアログボックスの下部に表示されます。



5 フィールド #NAME-END を選択し、［インポート］ボタンを選択します。

［データフィールドのインポート］ダイアログボックスの［キャンセル］ボタンのラベルが［終了］になります。

6 ［終了］ボタンを選択して、［データフィールドのインポート］ダイアログボックスを閉じます。

データフィールドがマップの左上に表示されます。 データフィールドは "X" 文字で埋められます。 ハンドルは、データフィールドが選択されていることを示します。

7 終了名用として前に定義したテキストフィールドの右に、データフィールドを移動します。これを行うには、データフィールドにマウスポインタを置き、マウスボタンを押し続けてマウスを目的の位置にドラッグしてから、マウスボタンを放します。

データフィールドの名前および属性の指定

次の説明は、手動で定義した開始名のデータフィールドに対してのみ適用されます。 インポートした終了名のデータフィールドには適用されません。ユーザー定義変数用に新しいデータフィールドを作成すると、Naturalによってフィールド名が割り当てられます。 このフィールド名には数値が含まれます。 新しく作成したフィールドの名前を、プログラムで定義された変数名に調整する必要があります。

同じ名前 #NAME-START および #NAME-END がプログラムで使用されていることを確認します。 これらのフィールドの出力（ユーザー入力）は、プログラム内の対応するユーザー定義変数に渡されます。

また、#NAME-START および #NAME-END の両方に同じ属性が定義されていることも確認します。

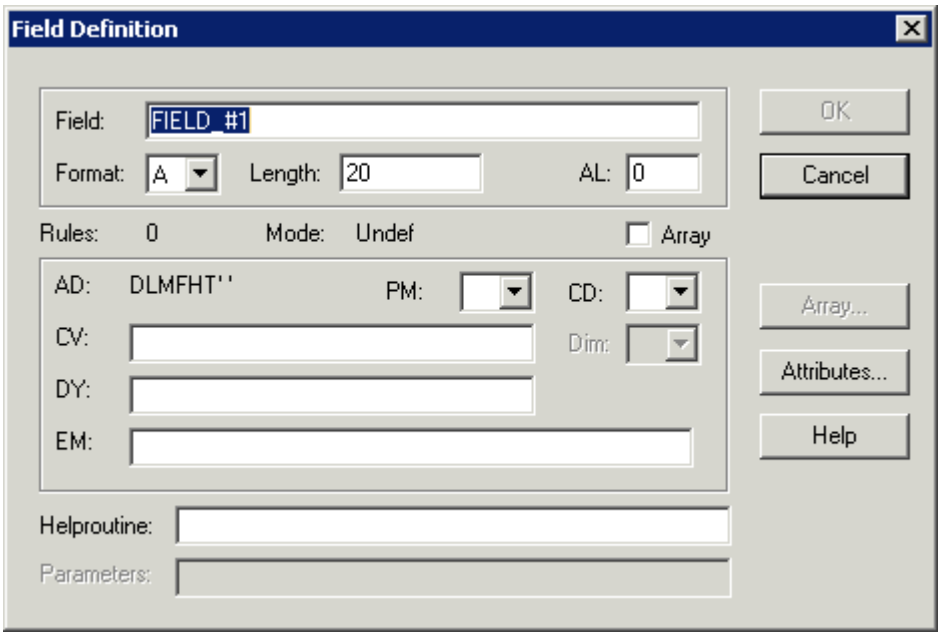
手順 6.7. データフィールドの名前と属性を定義するには

1 開始名のデータフィールドを選択し、コンテキストメニューから［定義］を選択します。

Or:

データフィールドをダブルクリックします。

［フィールドの定義］ダイアログボックスが表示されます。



［フィールド］テキストボックスに、Natural によって割り当てられたフィールド名 #FIELD_#1 が表示されます。

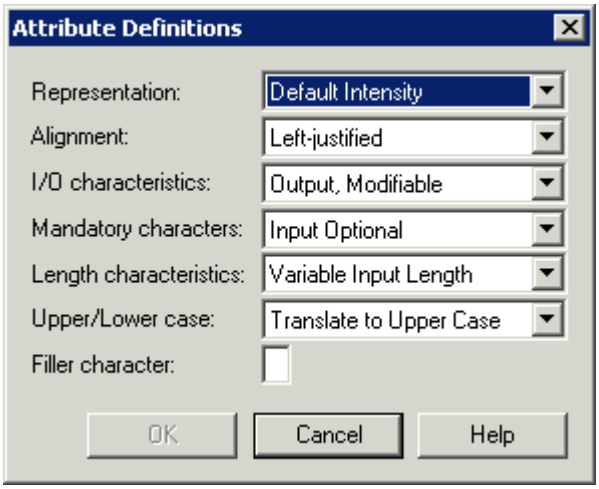
2 ［フィールド］テキストボックスに「#NAME-START」と入力します。

フォーマットは **A** である必要があります。 これは、デフォルトで選択されています。

3 ［長さ］テキストボックスに、「20」と入力します（フィールドの長さが異なる場合）。

4 ［属性］ボタンを選択します。

［属性の定義］ダイアログボックスが表示されます。



5 ［**I/O**特性］ドロップダウンリストボックスで［出力（変更可）］が選択されていることを確認します。

これにより、変更可能な出力フィールドとしてフィールドが定義されます。

6 ［大文字／小文字］ドロップダウンリストボックスで［英大文字に変換］が選択されていることを確認します。

これにより、ユーザーが名前を小文字で入力できるようになります。 これまでは、名前をすべて大文字で指定した場合にのみ、デモデータベースで検出されていました。

7 ［フィラー文字］テキストボックスに、下線（_）文字を入力します。

空白文字がデフォルトの充填文字として定義されます。 したがって、このフィールドに下線を入力する前に、空白を削除する必要があります。

充填文字を使用して、マップ内の入力フィールドの空の位置を埋めます。これにより、ユーザーは入力時にフィールドの正確な位置と長さを確認することができます。

8 ［**OK**］ボタンを選択して、［属性の定義］ダイアログボックスを閉じます。

9 ［**OK**］ボタンを選択して、［フィールドの定義］ダイアログボックスを閉じます。

10 終了名のデータフィールドについても、上記の手順を繰り返します。 フィールド名（#NAME-END）、フォーマット（A）、長さ（20）が正しく定義されていることを確認します。 #NAME-START と同じ属性が使用されていることを確認します。

システム変数の追加

Natural システム変数には、現在のライブラリ、ユーザー、日時など、現在の Natural セッションに関する情報が含まれます。 これらは、Natural プログラム内のどこからでも参照できます。 システム変数はすべてアスタリスク（*）で始まります。

日時のシステム変数をマップに追加します。 プログラムが実行されると、現在の日時がマップに表示されます。

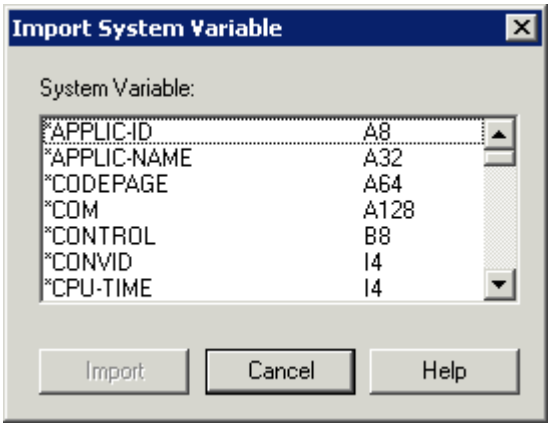
手順 6.8. システム変数を追加するには

1 ［挿入］メニューの［インポート］>［システム変数］を選択します。

Or:

次のツールバーボタンを選択します。

［システム変数のインポート］ダイアログボックスが表示されます。



2 `*DAT4I` までスクロールして選択します。

3 `*TIMX` までスクロールし、Ctrl キーを押して選択します。

4 ［インポート］ボタンを選択して、選択した変数をインポートします。

［システム変数のインポート］ダイアログボックスの［キャンセル］ボタンのラベルが［終了］になります。

5 ［終了］ボタンを選択して、ダイアログボックスを閉じます。

両方のシステム変数が、マップの左上に配置されました。

6 **TT:TT:TT**（時刻のシステム変数）を選択し、先頭行の最後に移動します。

Note: 行の最後を表示するには、マップエディタウィンドウのサイズを変更する必要がある場合があります。

マップのテスト

マップが意図したとおりに動作するかどうかをテストします。

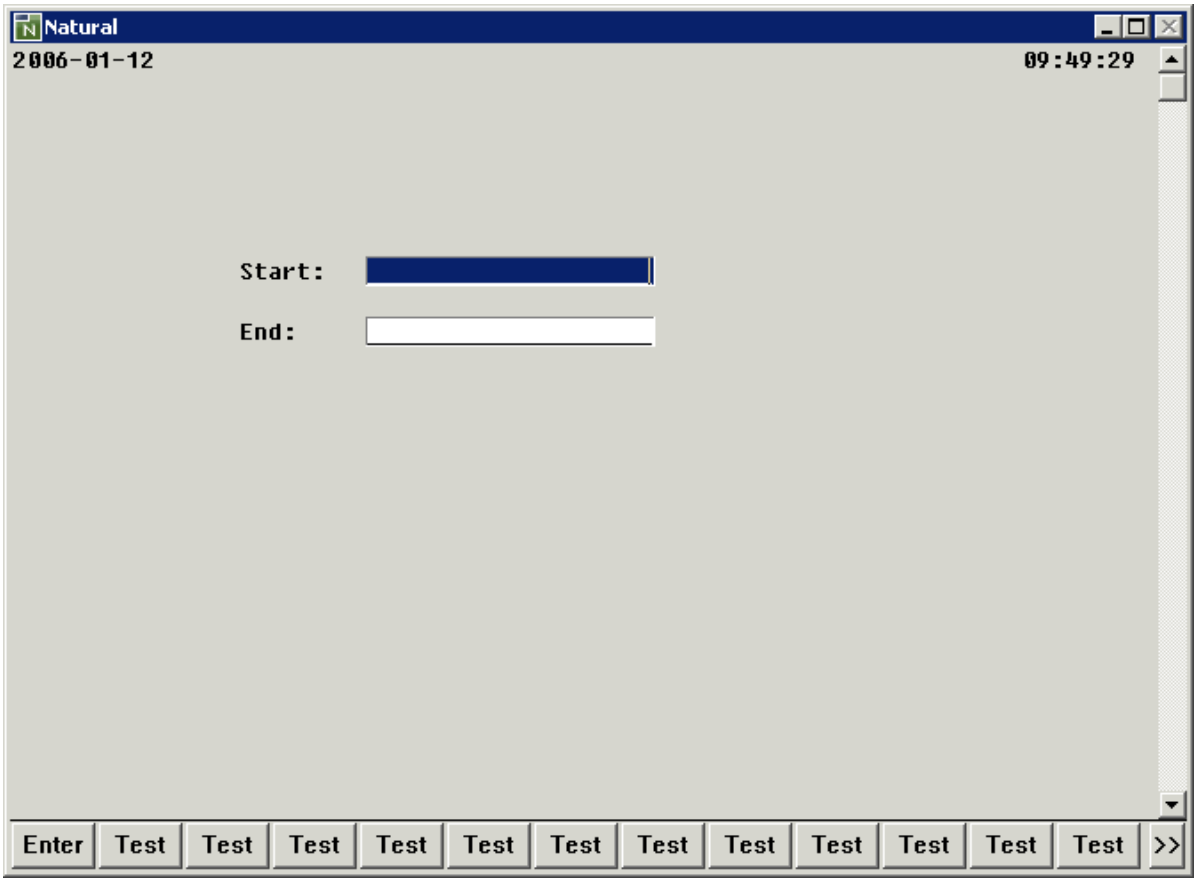
手順 6.9. マップをテストするには

1 ［オブジェクト］メニューの［**Run**］を選択します。

Or:

次のツールバーボタンを選択します。

次の出力が表示されます。



Note: 右上にある時刻を表示するため、出力ウィンドウのサイズを変更する必要がある場合があります。

開始名の入力フィールドはマップの最初の入力フィールドであるため、自動的に選択されます。 どちらの入力フィールドにも、充填文字が含まれています。

Note: 挿入モードで操作する場合、テキストを入力する前に充填文字を削除する必要があります。 上書きモードでは、この操作は不要です。

2 Enter キーを押して、マップエディタに戻ります。

マップの格納

マップのテストが正常に行われたら、プログラムで検出できるようにするため、マップを格納する必要があります。

手順 6.10. マップを Stow するには

1 マップの格納はプログラムの場合と同じ方法で行います。

2 マップ名の指定を求められたら、「MAP01」と入力します。

ライブラリワークスペースで、**Maps** という名前の新しいノードが **TUTORIAL** ノードの下に表示されます。 このノードには、格納したマップが含まれています。

後で修正を加えるために、マップエディタを開いたままにしておきます。

プログラムからのマップの起動

マップの格納が完了したら、`WRITE` または `INPUT` ステートメントを使用して Natural プログラムから起動できます。

手順 6.11. プログラムからマップを起動するには

1 プログラムエディタに戻ります。

（マップエディタウィンドウを前に最大表示したなどの理由で）プログラムエディタが表示されない場合は、［ウィンドウ］メニュー下部にある `PGM01` に対応するコマンドを選択することにより、開いているプログラムエディタウィンドウに戻ることができます。

また、ライブラリワークスペースでプログラム `PGM01` をダブルクリックする（キーボードで操作している場合は、プログラムを選択して Enter キーを押す）こともできます。 プログラムが前に閉じられている場合は、再び開きます。 プログラムがバックグラウンドで開いたままになっている場合は、エディタウィンドウが手前に表示されます。

2 前に定義した `INPUT` 行を次の行で置き換えます。

```
INPUT USING MAP 'MAP01'
```

これにより、設計したマップが起動されます。

マップをユーザー定義変数と区別するため、マップ名を一重引用符で囲む必要があります。

プログラムは次のようになります。

```
DEFINE DATA
LOCAL
  1 #NAME-START        (A20) INIT <"ADKINSON">
  1 #NAME-END          (A20) INIT <"BENNETT">
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
INPUT USING MAP 'MAP01'
*
READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
END
```

3 プログラムを実行します。

マップが表示されます。

4 プログラムエディタに戻るまで繰り返し Enter キーを押すか、Esc キーを押します。。

5 プログラムを格納します。

終了名を常に使用するための操作

プログラムをコーディングしても、終了名が指定されていなければデータは検出されません。

開始名および終了名の初期値を削除すると、これらの名前をユーザーが常に指定する必要があります。 ユーザーが終了名を指定しなくても、終了名が常に使用されるようにするため、対応するステートメントを追加します。

手順 6.12. 終了名を使用するには

1 DEFINE DATAブロックで、フィールド#NAME-STARTおよび#NAME-ENDのデフォルト値（INIT）を削除します。対応する行は次のようになります。

```
1 #NAME-START        (A20)
1 #NAME-END          (A20)
```

2 INPUT USING MAP 'MAP01'の下に、次の行を挿入します。

```
IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
END-IF
```

#NAME-ENDフィールドが空白になっている（ユーザーが終了名を入力しなかった）場合は、開始名が自動的に終了名として使用されます。

Note: ステートメントMOVE #NAME-START TO #NAME-ENDを使用する代わりに、ASSIGNまたはCOMPUTEステートメントの次の変形#NAME-END := #NAME-STARTを使用することもできます。

プログラムは次のようになります。

```
DEFINE DATA
LOCAL
  1 #NAME-START        (A20)
  1 #NAME-END          (A20)
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
INPUT USING MAP 'MAP01'
*
IF #NAME-END = ' ' THEN
  MOVE #NAME-START TO #NAME-END
END-IF
*
READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
```

```
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
END
```

3 プログラムを実行します。

4 結果のマップで、開始名の入力を求めるフィールドに「JONES」と入力し、Enter キーを押します。

Note: このフィールドには［英大文字に変換］が指定されているため、名前を小文字で入力することもできます。

結果のリストには、"Jones" という名前の従業員のみが表示されます。

5 Enter キーを押して、プログラムエディタに戻ります。

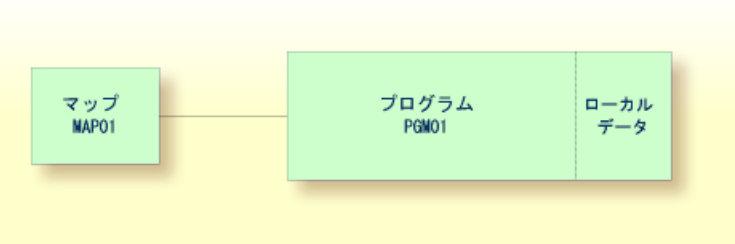
6 プログラムを格納します。

次の演習「*ループおよびラベル*」に進みます。

7 ループおよびラベル

ループおよびラベルを追加してプログラムを強化します。

以下の演習を完了すると、サンプルのアプリケーションを構成するモジュールは前のchapterと同じになります。



このchapterには次の演習が含まれています。

反復使用の許可

現時点では、マップが表示され、リストが表示された時点でプログラムは終了します。 プログラムを再スタートしなくても新しい従業員リストを直ちに表示できるようにするため、対応するプログラムコードを REPEAT ループに挿入します。

▶手順 7.1. 繰り返しループを定義するには

1 END-DEFINE の下に次の行を挿入します。

```
RP1. REPEAT
```

REPEATは、繰り返しループの開始を定義します。 RP1.は、繰り返しループを終了するときに使用されるラベルです（以下で定義）。

2 END ステートメントの前に次の行を挿入して、繰り返しループの終了を定義します。

```
END-REPEAT
```

3 INPUT USING MAP 'MAP01' の下に、次の行を挿入します。

```
IF #NAME-START = '.' THEN
  ESCAPE BOTTOM (RP1.)
END-IF
```

IF ステートメントは、END-IF で終了する必要があり、#NAME-START フィールドの内容をチェックします。このフィールドに「.」（ピリオド）を入力すると、ループの終了に ESCAPE BOTTOM ステートメントが使用されます。 ループの後にある最初のステートメント（この場合は END）から処理が続行します。

ループにラベル（ここでは RP1）を割り当てると、このループを ESCAPE BOTTOM ステートメントで参照できます。 ループはネストすることができるため、どのループを終了するか指定する必要があります。 指定がない場合は、最も内側にあるアクティブなループが終了します。

プログラムは次のようになります。

```
DEFINE DATA
LOCAL
  1 #NAME-START        (A20)
  1 #NAME-END          (A20)
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
```

```
  END-READ
*
END-REPEAT
*
END
```

Note: 読みやすくするため、REPEAT ループのコンテンツはインデントされています。

4 プログラムを実行します。

5 結果のマップで、開始名の入力を求めるフィールドに「JONES」と入力し、Enter キーを押します。

結果のリストに、"Jones" という名前の従業員が表示されます。 Enter キーを押します。 REPEAT ループにより、マップが再表示されます。 終了名にも「JONES」と入力されていることがわかります。

6 マップを終了するには、開始名の入力を求めるフィールドに「.」（ピリオド）を入力し、Enter キーを押します。 このフィールドに表示されたままになっている名前の残りの文字は、必ず削除してください。

7 プログラムを格納します。

情報が見つからないことを示すメッセージの表示

データベースで検出できない開始名をユーザーが入力したときに表示するメッセージを定義します。

手順 7.2. 指定の従業員が見つからない場合に表示するメッセージを定義するには

1 READ ステートメントを含む行にラベル RD1. を追加します。行は次のようになります。

```
RD1. READ EMPLOYEES-VIEW BY NAME
```

2 END-READ の下に次の行を挿入します。

```
IF *COUNTER (RD1.) = 0 THEN
   REINPUT 'No employees meet your criteria.'
END-IF
```

READ ループで検出されたレコード数をチェックするため、システム変数 *COUNTER を使用します。 この変数の値が 0 である（指定した名前の従業員が見つからない）と、REINPUT ステートメントで定義されたメッセージがマップ下部に表示されます。

READ ループを識別するため、ループにラベル（ここでは RD1.）を割り当てます。 データベースにアクセスする複雑なプログラムでは多くのループが含まれている場合があるため、参照するループを指定する必要があります。

プログラムは次のようになります。

```
DEFINE DATA
LOCAL
  1 #NAME-START        (A20)
  1 #NAME-END          (A20)
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT
*
END
```

3 プログラムを実行します。

4 結果のマップで、デモデータベースで定義されていない開始名（「XYZ」など）を入力し、Enter キーを押します。

マップにメッセージが表示されます。

5 マップを終了するには、開始名の入力を求めるフィールドに「.」（ピリオド）を入力し、Enter キーを押します。このフィールドに表示されたままになっている名前の残りの文字は、必ず削除してください。

Or:

Esc キーを押します。

6 プログラムを格納します。

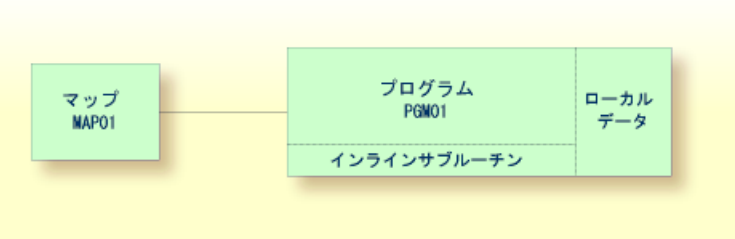
次の演習「*インラインサブルーチン*」に進みます。

8 インラインサブルーチン

Naturalでは、プログラム内で直接定義されるインラインサブルーチンと、プログラム外で別のオブジェクトとして格納される外部サブルーチン（このチュートリアルで後述）という2種類のサブルーチンが区別されます。

#MARK という名前のユーザー定義変数にアスタリスク（*）を移動するインラインサブルーチンをプログラムに追加します。 このサブルーチンは、従業員が20日以上の休暇を取った場合に起動されます。

以下の演習を完了すると、サンプルのアプリケーションは次の構造になります。



このchapterには次の演習が含まれています。

インラインサブルーチンの定義

サブルーチンをプログラムに追加します。

手順 8.1. サブルーチンを定義するには

1 ユーザー定義変数 #NAME-END の下に次の行を挿入します。

```
1 #MARK              (A1)
```

この変数はサブルーチンで使用されます。 したがって、最初に定義しておく必要があります。

2 サブルーチンを定義するため、END ステートメントの前に次の行を挿入します。

```
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
```

このサブルーチンが実行されると、アスタリスク（*）が #MARK に移動します。

Note: ステートメント MOVE '*' TO #MARK の代わりに、ASSIGN または COMPUTE ステートメントの次の変形 #MARK := '*' を使用することもできます。

3 DISPLAY ステートメントを次のように変更します。

```
DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK
```

これにより、出力に新しい列が表示されます。見出しは">=20"です。該当する従業員が20日以上の休暇を取った場合、列にアスタリスク（*）が表示されます。

インラインサブルーチンの実行

インラインサブルーチンの定義が完了したので、インラインサブルーチンを実行するコードを指定できます。

手順 8.2. インラインサブルーチンを実行するには

1 DISPLAY ステートメントの下に次の行を挿入します。

```
IF LEAVE-DUE >= 20 THEN
  PERFORM MARK-SPECIAL-EMPLOYEES
ELSE
  RESET #MARK
END-IF
```

20日以上の休暇を取った従業員が検出されると、MARK-SPECIAL-EMPLOYEES という名前の新しいサブルーチンが実行されます。 従業員の休暇が 20 日未満の場合、#MARK のコンテンツは空白にリセットされます。

プログラムは次のようになります。

```
DEFINE DATA
LOCAL
  1 #NAME-START        (A20)
  1 #NAME-END          (A20)
  1 #MARK              (A1)
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
```

```
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    IF LEAVE-DUE >= 20 THEN
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
*
END
```

2 プログラムを実行します。

3 結果のマップで、「JONES」と入力して Enter キーを押します。

従業員のリストに列が追加されて表示されます。

4 Esc キーを押して、出力ウィンドウを閉じます。

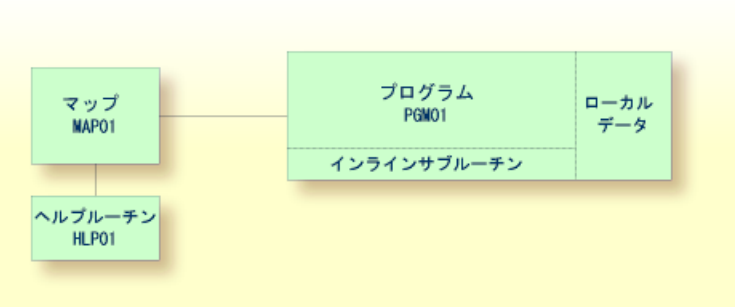
5 プログラムを格納します。

次の演習「*処理ルールとヘルプルーチン*」に進みます。

9 処理ルールとヘルプルーチン

処理ルールとヘルプルーチンは、マップ内のフィールドに対して定義されます。

以下の演習を完了すると、サンプルのアプリケーションは次のモジュールで構成されます（処理ルールは別のモジュールとして定義できず、常にマップの一部になります）。



このchapterには次の演習が含まれています。

処理ルールの定義

ユーザーが開始名を指定せずに Enter キーを押した場合に表示するメッセージを定義します。

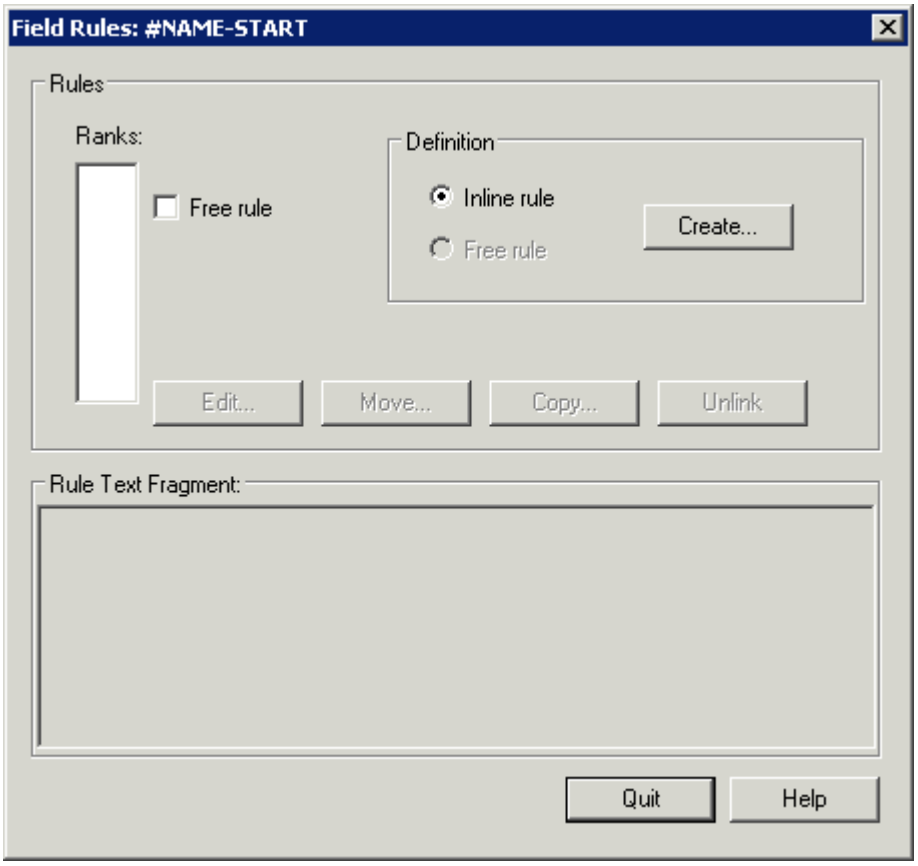
▶手順 9.1. 処理ルールを定義するには

1 マップエディタに戻ります。

2 開始名の入力フィールドを選択します。

3 コンテキストメニューから、［ルール］を選択します。

フィールド #NAME-START 用の［フィールド処理ルール］ダイアログボックスが表示されます。



4 ［生成］ボタンを選択します。

空のエディタウィンドウが表示されます。

5 次の処理ルールを入力します。

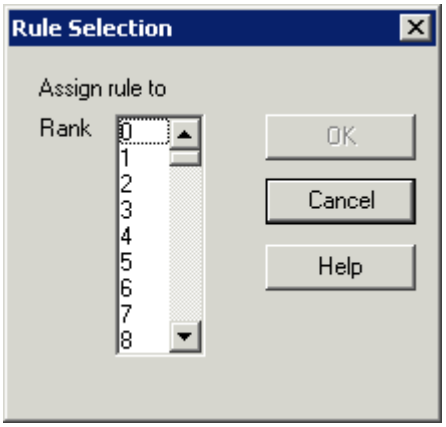
```
IF & = ' ' THEN
  REINPUT 'Please enter a starting name.'
  MARK *&
END-IF
```

処理ルール内のアンパサンド（&）は、フィールド名にダイナミックに置き換えられます。この場合は、#NAME-START に置き換えられます。#NAME-START が空白の場合は、REINPUT ステートメントで定義されたメッセージが表示されます。

MARK は REINPUT ステートメントのオプションです。 構文は MARK *fieldname* です。 MARK は、REINPUT ステートメントの実行時にカーソルが置かれるフィールドを指定します。 この場合は、カーソルが #NAME-START フィールドに置かれます。

6 エディタウィンドウの内容を保存します。

［ルール選択］ダイアログボックスが表示されます。



7 ［ランク］リストボックスで［**1**］を選択し、［**OK**］ボタンを選択します。

ランクにより、異なるフィールドに対するルールの処理順序が定義されます。 ランク 1 のルールはすべて最初に処理され、その後にランク 2 などが続きます。

8 処理ルールを入力したエディタウィンドウを閉じます。

9 マップをテストします。

10 結果の出力で、任意の開始名を入力して Enter キーを押します。

出力ウィンドウが閉じます。

11 マップをもう一度テストします。 名前を入力せずに、Enter キーを押します。

処理ルールで定義したメッセージが、マップに表示されます。

12 出力ウィンドウを終了するには、開始名の入力を求めるフィールドに「.」（ピリオド）を入力し、Enter キーを押します。

13 マップを Stow します。

ヘルプルーチンの定義

ヘルプルーチンは、開始名の入力フィールドにカーソルが置かれているときにユーザーがヘルプキーを押すと表示されます。

まず、ヘルプルーチンを定義してから、それを特定のフィールドに関連付けます。

手順 9.2. ヘルプルーチンを作成するには

1 ライブラリワークスペースで、プログラムも含まれているライブラリ（**TUTORIAL** ノード）を選択します。

2 コンテキストメニューから、［新規作成］>［ヘルプルーチン］を選択します。

空のエディタが表示されます。

3 次のように入力します。

```
WRITE 'Type the name of an employee'
END
```

4 ヘルプルーチンを格納し HLP01ます。

［名前を付けて **Stow**］ダイアログボックスが表示されます。

5 ヘルプルーチンの名前として「HLP01」と入力します。

6 ［**OK**］ボタンを選択します。

ライブラリワークスペースで、**Helproutines** という名前の新しいノードが **TUTORIAL** ノードの下に表示されます。 このノードには、格納したヘルプルーチンが含まれています。

7 ヘルプルーチンを入力したエディタウィンドウを閉じます。

手順 9.3. ヘルプルーチンをマップ上のフィールドに関連付けるには

1 マップエディタに戻ります。

2 開始名のデータフィールドを選択します。

3 コンテキストメニューから、［定義］を選択します。

Or:

データフィールドをダブルクリックします。

［フィールドの定義］ダイアログボックスが表示されます。

4 ［ヘルプルーチン］テキストボックスで「'HLP01'」（一重引用符も含めます）と入力します。

これは、ヘルプルーチンの保存に使用した名前です。

5 ［**OK**］ボタンを選択します。

6 マップをテストします。

7 結果の出力で、開始名の入力フィールドに疑問符（?）を入力します。

定義したヘルプテキストが別のウィンドウに表示されます。

8 Enter キーを押して、このウィンドウを閉じます。

9 マップを終了するには、開始名の入力を求めるフィールドに「.」（ピリオド）を入力し、Enter キーを押します。

10　マップを Stow します。

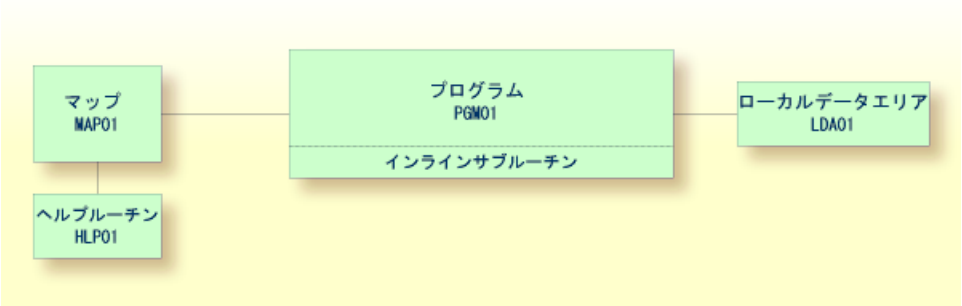
11　マップエディタウィンドウを閉じます。

次の演習「*ローカルデータエリア*」に進みます。

10 ローカルデータエリア

現時点では、プログラムで使用するフィールドは、プログラム内の DEFINE DATA ステートメントで定義されています。 このフィールド定義をプログラム外のローカルデータエリア（LDA）に配置し、プログラムの DEFINE DATA ステートメントを使用して、ローカルデータエリアを名前で参照することもできます。 再利用および明確なアプリケーション構造という観点から見ると、通常、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が推奨されます。

情報を DEFINE DATA ステートメントからローカルデータエリアに再配置します。 以下の演習を完了すると、サンプルのアプリケーションは次のモジュールで構成されます。



このchapterには次の演習が含まれています。

ローカルデータエリアの作成

必須フィールドを指定するデータエリアエディタを起動します。

手順 10.1. ローカルデータエリアを作成するには

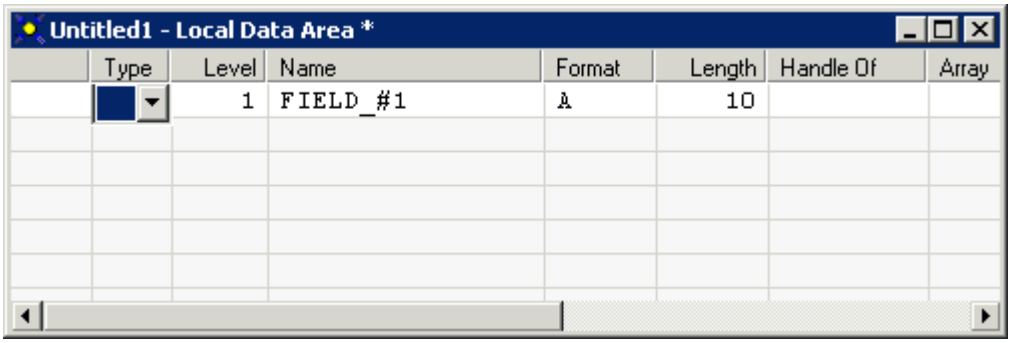
1 ライブラリワークスペースで、プログラムも含まれているライブラリ（**TUTORIAL** ノード）を選択します。

2 コンテキストメニューから、［新規作成］>［ローカルデータエリア］を選択します。

Or:

次のツールバーボタンを選択します。

エディタウィンドウが表示されます。



レベル、名前、フォーマット、および長さは、最初の行で自動的に事前設定されます。

データフィールドの定義

次のフィールドを定義します。

レベル	名前	フォーマット	長さ
1	#NAME-START	A	20
1	#NAME-END	A	20
1	#MARK	A	1

これらは、DEFINE DATA ステートメントで前に定義したユーザー定義変数です。

データフィールドは2つの方法で定義できます。エディタウィンドウで手動で定義する場合は、データフィールドに正しいフォーマットと長さを定義する必要があります。［インポート］コマンドを使用すると、データフィールドをリストから選択するだけで、正しいフォーマットと長さが自動的に使用されます。 これらの 2 つの方法について、以下で説明します。

データエリアエディタの［挿入］ツールバーにある次のツールバーボタンを設定すると、挿入位置が示されます。 ツールバーボタンが押されているように見える場合、選択したフィールドの後に新しいフィールドが挿入されます（このチュートリアルでは、この状態を前提にしています）。そうでない場合は、選択したフィールドの前に新しいフィールドが挿入されます。

手順 10.2. エディタウィンドウでデータフィールドを手動で定義するには

1 最初の行の［レベル］列にプリセット値 "1" が含まれていることを確認します。
2 最初の行の［名前］列のプリセット値を "#NAME-START" に変更します。
3 最初の行の［フォーマット］列にプリセット値 "A" が含まれていることを確認します。
4 最初の行の［長さ］列のプリセット値を "20" に変更します。

手順 10.3. プログラムからデータフィールドをインポートするには

1 エディタで、#NAME-START 用に作成した行を選択します。

Note: 必要に応じて、Enter キーまたは Esc キーを押して、単一セル選択モードから全列選択モードに切り替えます。

2 コンテキストメニューから、［インポート］を選択します。

Or:

次のツールバーボタンを選択します。

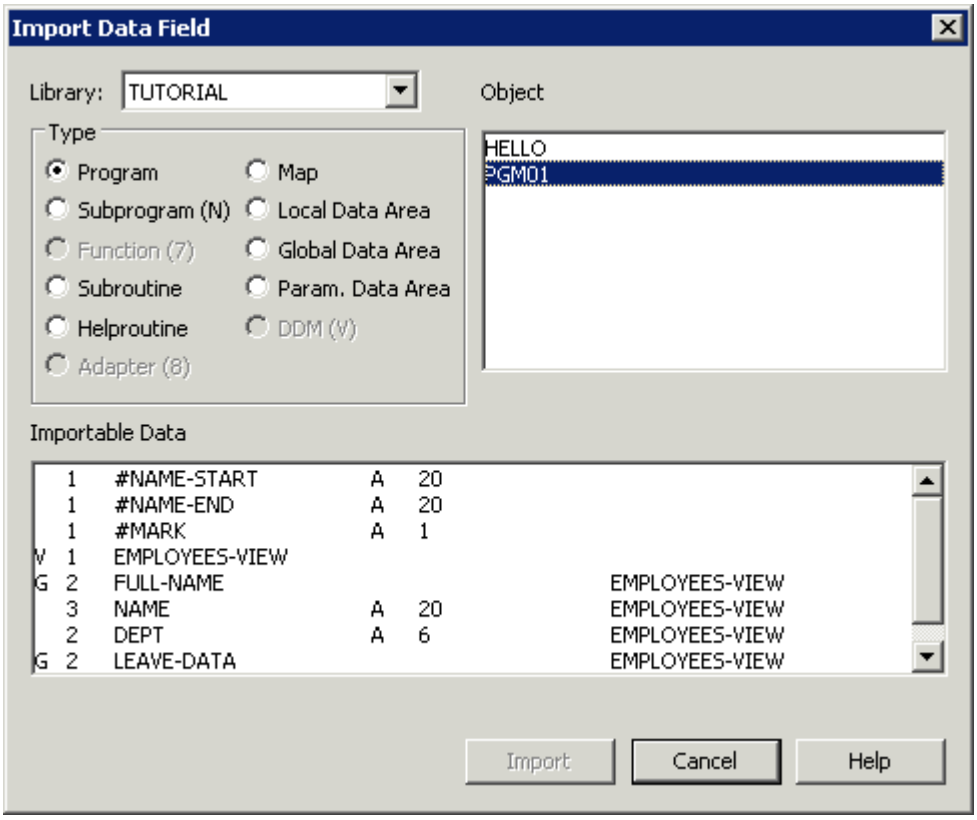
［データフィールドのインポート］ダイアログボックスが表示されます。

3 ［ライブラリ］ドロップダウンリストボックスで**TUTORIAL**が選択されていることを確認します。
4 ［プログラム］チェックボックスをオンにします。

ライブラリで現在定義されているすべてのプログラムが、［オブジェクト］リストボックスに表示されます。

5 PGM01 という名前のプログラムを選択します。

インポートできるデータフィールドがダイアログボックスの下部に表示されます。



6 Ctrl キーを押して、［インポータブルデータ］リストボックスで次のフィールドを選択します。

#NAME-END
#MARK

7 ［インポート］ボタンを選択します。

［データフィールドのインポート］ダイアログボックスの［キャンセル］ボタンのラベルが［終了］になります。

8 ［終了］ボタンを選択して、［データフィールドのインポート］ダイアログボックスを閉じます。

フィールド #NAME-END および #MARK が、エディタウィンドウで #NAME-START フィールドの下に表示されます。

DDM からの必須データフィールドのインポート

プログラムの DEFINE DATA ステートメントで前に定義したデータフィールドをインポートします。 フィールドは、Natural データビューからデータエリアエディタに直接読み込まれます。データビューは、データ定義モジュール（DDM）で定義されたデータベースフィールドを参照します。

データエリアエディタで、インポートしたデータフィールドは、現在選択されているデータフィールドの下に挿入されます。

手順 10.4. DDM からデータフィールドをインポートするには

1 エディタで、#MARK を含む行を選択します。

2 コンテキストメニューから、［インポート］を選択します。

［データフィールドのインポート］ダイアログボックスが表示されます。

3 ［ライブラリ］ドロップダウンリストボックスから［**SYSEXDDM**］を選択します。

［**DDM**］オプションボタンが選択されています。

4 ［オブジェクト］リストボックスで、**EMPLOYEES** という名前のサンプル DDM を選択します。

5 Ctrl キーを押して、［インポータブルデータ］リストボックスで次のフィールドを選択します。

```
PERSONNEL-ID
FULL-NAME
NAME
DEPT
LEAVE-DATA
LEAVE-DUE
```

Note: フィールド PERSONNEL-ID は、後でサブプログラムを作成するときに使用します。

6 ［インポート］ボタンを選択します。

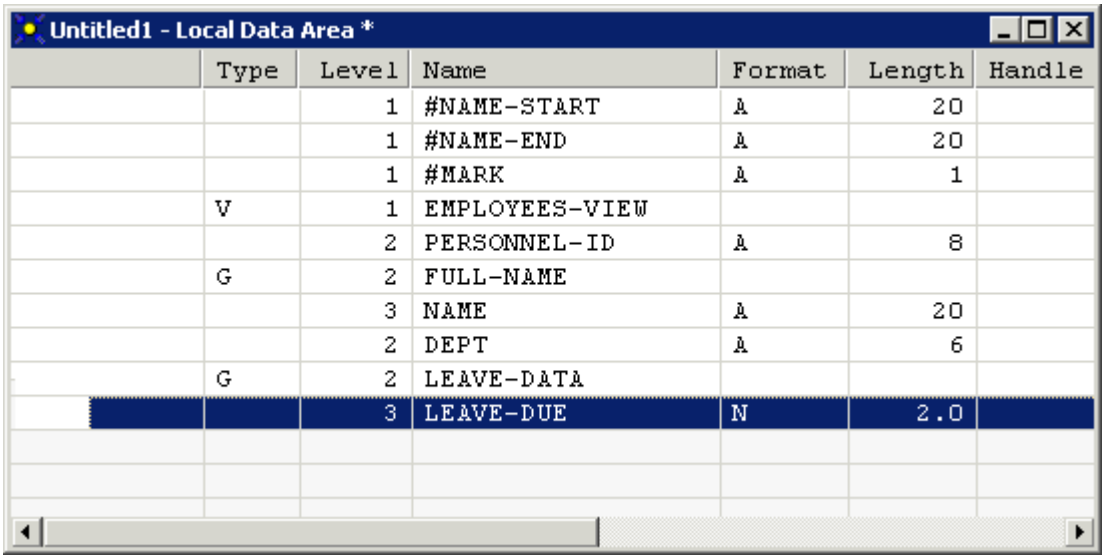
［ビューの定義］ダイアログボックスが表示されます。

7 ビューに対して前に定義した名前（EMPLOYEES-VIEW）を指定します。

8 ［**OK**］ボタンを選択します。

9 ［終了］ボタンを選択して、［データフィールドのインポート］ダイアログボックスを閉じます。

ローカルデータエリアが次のように表示されます。



	Type	Level	Name	Format	Length	Handle
		1	#NAME-START	A	20	
		1	#NAME-END	A	20	
		1	#MARK	A	1	
	V	1	EMPLOYEES-VIEW			
		2	PERSONNEL-ID	A	8	
	G	2	FULL-NAME			
		3	NAME	A	20	
		2	DEPT	A	6	
	G	2	LEAVE-DATA			
		3	LEAVE-DUE	N	2.0	

Type 列は、変数タイプを示します。 ビューは "V" で示され、各グループは "G" で示されます。

Note: データエリアエディタオプションで［展開／圧縮］チェックボックスがオンになっていると、展開／圧縮トグルが（ビューおよび各グループの）最初の列に表示されます。

10 ローカルデータを格納します。

11 ローカルデータエリア名の指定を求められたら、「LDA01」と入力します。

ライブラリワークスペースで、**Local Data Areas** という名前の新しいノードが **TUTORIAL** ノードの下に表示されます。 このノードには、格納したローカルデータエリアが含まれています。

プログラムからのローカルデータエリアの参照

ローカルデータエリアを格納すると、Natural プログラムから参照できます。

定義したローカルデータエリアを使用するため、プログラムの `DEFINE DATA` ステートメントを変更します。

データエリアエディタをバックグラウンドで開いたままにします。

手順 10.5. プログラムでローカルデータエリアを使用するには

1 プログラムエディタに戻ります。

2 DEFINE DATA ステートメントで、LOCAL と END-DEFINE 間にある変数をすべて削除します。

3 LOCAL 行を次のように変更して、ローカルデータエリアへの参照を追加します。

```
LOCAL USING LDA01
```

プログラムは次のようになります。

```
DEFINE DATA
  LOCAL USING LDA01
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    IF LEAVE-DUE >= 20 THEN
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
```

```
END-SUBROUTINE
*
END
```

4 プログラムを実行します。

5 前（DEFINE DATA ステートメントでローカルデータエリアを参照していない場合）と同じ結果になることを確認するため、開始名に「JONES」と入力して、Enter キーを押します。

6 Esc キーを押して、出力ウィンドウを閉じます。

7 プログラムを格納します。

次の演習「グローバルデータエリア」に進みます。

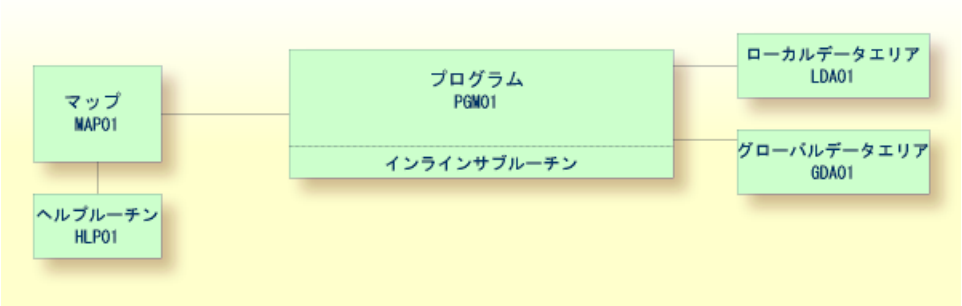
11 グローバルデータエリア

グローバルデータエリア（GDA）で定義されたデータは、複数のプログラム、外部サブルーチン、およびヘルプルーチンで共有することができます。

グローバルデータエリアにあるデータ要素の値に加えた変更は、そのグローバルデータエリアを参照するすべての Natural オブジェクトに影響を与えます。 したがって、グローバルデータエリアのソースを変更した場合は、そのグローバルデータエリアを参照する、作成済みのすべての Natural オブジェクトをもう一度格納する必要があります。オブジェクトを格納する順序は重要です。まず、グローバルデータエリアを格納してから、プログラムを格納する必要があります。この順序を逆にすると、グローバルデータエリアにある新しい要素を検出できなくなるため、プログラムを格納できなくなります。

プログラムおよび後で作成する外部サブルーチンで共有するグローバルデータエリアを作成します。 グローバルデータエリアのベースとして、作成済みのローカルデータエリアの一部の情報を使用します。

以下の演習を完了すると、サンプルのアプリケーションは次のモジュールで構成されます。



このchapterには次の演習が含まれています。

既存のローカルデータエリアからのグローバルデータエリアの作成

既存のデータエリアを編集し、それを異なる名前および異なるタイプで保存することにより、既存のデータエリアから新しいデータエリアを作成できます。 元のデータエリアは変更されないまま残り、新しいデータエリアを編集できます。 フィールド #NAME-START および #NAME-END はグローバルデータエリアには必要ないため、削除します。

Note: また、［オブジェクト］メニューの［新規作成］>［グローバルデータエリア］を選択して、グローバルデータエリアを作成することもできます。

手順 11.1. グローバルデータエリアを作成するには

1 ローカルデータエリアに戻ります。

2 ［オブジェクト］メニューの［名前を付けて保存］を選択します。

［名前をつけて保存］ダイアログボックスが表示されます。

3 グローバルデータエリアの名前として「GDA01」と指定します。

4 プログラムも含むライブラリ（**TUTORIAL** ノード）が選択されていることを確認します。

5 ［タイプ］ドロップダウンリストボックスの［グローバル］を選択します。

6 ［**OK**］ボタンを選択します。

新しい名前とタイプがエディタウィンドウのタイトルバーに表示されます。ライブラリワークスペースで、新しいグローバルデータエリアが **Global Data Areas** ノードに表示されます。

7 Ctrl キーを押して、次のフィールドを選択します。

```
#NAME-START
#NAME-END
```

8 コンテキストメニューから、［削除］を選択します。

Or:

Delete キーを押します。

グローバルデータエリアは次のようになります。

GDA01 [TUTORIAL] - Global Data Area *

	Type	Level	Name	Format	Length	Handle
		1	#MARK	A	1	
	V	1	EMPLOYEES-VIEW			
		2	PERSONNEL-ID	A	8	
	G	2	FULL-NAME			
		3	NAME	A	20	
		2	DEPT	A	6	
	G	2	LEAVE-DATA			
		3	LEAVE-DUE	N	2.0	

9 グローバルデータエリアを格納します。

プログラムおよび後で定義する外部サブルーチンで、グローバルデータエリアを検出できるようになりました。

10　グローバルデータエリアのエディタウィンドウを閉じます。

ローカルデータエリアへの適合

グローバルデータエリアに含まれるフィールドは、ローカルデータエリアに必要ありません。したがって、#NAME-START および #NAME-END を除くすべてのフィールドをローカルデータエリアから削除します。

手順 11.2. フィールドを削除するには

1　ライブラリワークスペースでローカルデータエリア LDA01 を選択し、コンテキストメニューから［開く］を選択します。

Or:

ライブラリワークスペースで、ローカルデータエリア LDA01 をダブルクリックします。

2　#NAME-START および #NAME-END を除くすべてのフィールドを、結果のデータエリアエディタで選択します。

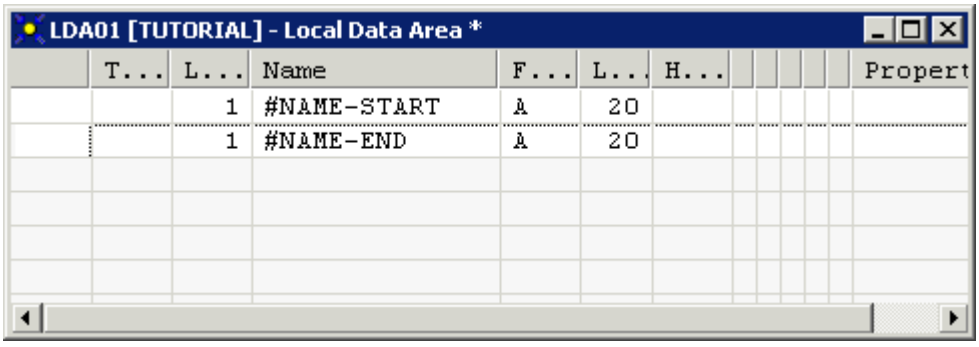
3　コンテキストメニューから、［削除］を選択します。

Or:

Delete キーを押します。

4　変更したローカルデータエリアを格納します。

ローカルデータエリアが次のように表示されます。



プログラムからのグローバルデータエリアの参照

グローバルデータエリアを格納すると、Natural プログラムから参照できます。

定義したグローバルデータエリアも使用するように、プログラムの DEFINE DATA ステートメントを変更します。

データエリアエディタをバックグラウンドで開いたままにします。

手順 11.3. プログラムでグローバルデータエリアを使用するには

1 プログラムエディタに戻ります。

2 LOCAL USING LDA01 の上の行に、次のように挿入します。

```
GLOBAL USING GDA01
```

グローバルデータエリアは、常にローカルデータエリアより先に定義する必要があります。そのようにしないと、エラーが発生します。

プログラムは次のようになります。

```
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    IF LEAVE-DUE >= 20 THEN
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
```

```
      RESET #MARK
    END-IF
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
*
END
```

3 プログラムを実行します。

4 前（DEFINE DATA ステートメントでグローバルデータエリアを参照していない場合）と同じ結果になることを確認するため、開始名に「JONES」と入力して、Enter キーを押します。

5 Esc キーを押して、出力ウィンドウを閉じます。

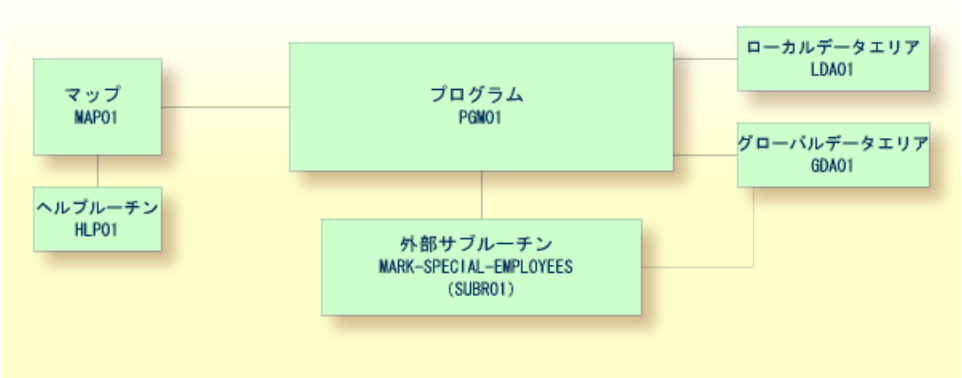
6 プログラムを格納します。

次の演習「*外部サブルーチン*」に進みます。

12 外部サブルーチン

これまでは、DEFINE SUBROUTINE ステートメントを使用して、サブルーチン MARK-SPECIAL-EMPLOYEES をプログラム内で定義していました。この演習では、サブルーチンを別のオブジェクトとしてプログラムの外部に定義します。

以下の演習を完了すると、サンプルのアプリケーションは次のモジュールで構成されます。



このchapterには次の演習が含まれています。

外部サブルーチンの作成

外部サブルーチンのコードを指定するエディタを起動します。

外部サブルーチンの DEFINE SUBROUTINE ステートメントは、プログラム内のインラインサブルーチンと同じ方法でコーディングします。

手順 12.1. 外部サブルーチンを作成するには

1 ライブラリワークスペースで、プログラムも含まれているライブラリ（**TUTORIAL** ノード）を選択します。

2 コンテキストメニューから、［新規作成］>［サブルーチン］を選択します。

空のエディタウィンドウが表示されます。

3 次のように入力します。

```
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
*
END
```

4 サブルーチンを格納します。

［名前を付けて **Stow**］ダイアログボックスが表示されます。

5 外部サブルーチンの名前として「SUBR01」と入力し、［**OK**］ボタンを選択します。

ライブラリワークスペースで、新しい外部サブルーチンが **Subroutines** ノードに表示されます。 論理ビューには、コード `MARK-SPECIAL-EMPLOYEES` で定義されたサブルーチンの名前が表示されます。 他のすべてのビューには、`SUBR01` という名前が表示されます。

6 外部サブルーチンを入力したエディタウィンドウを閉じます。

プログラムからの外部サブルーチンの参照

`PERFORM` ステートメントは、内部サブルーチンおよび外部サブルーチンの両方を呼び出します。内部サブルーチンがプログラム内で見つからないと、Naturalは同じ名前の外部サブルーチンを自動的に実行しようとします。 Naturalでは、サブルーチンコードで定義された名前（サブルーチン名）が検索され、サブルーチンの保存時に指定した名前（Naturalオブジェクト名）が検索されるわけではありません。

外部サブルーチンの定義が完了したので、インラインサブルーチン（外部サブルーチンと同じ名前を持つ）をプログラムから削除する必要があります。

手順 12.2. プログラムで外部サブルーチンを使用するには

1 プログラムエディタに戻ります。

2 次の行を削除します。

```
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
```

プログラムは次のようになります。

```
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    IF LEAVE-DUE >= 20 THEN
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK

  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT
*
END
```

3 プログラムを実行します。

4 開始名として「JONES」と入力し、Enter キーを押します。

結果のリストには、20 日以上の休暇を取った各従業員にアスタリスクが引き続き表示されます。

5 Esc キーを押して、出力ウィンドウを閉じます。

6 プログラムを格納します。

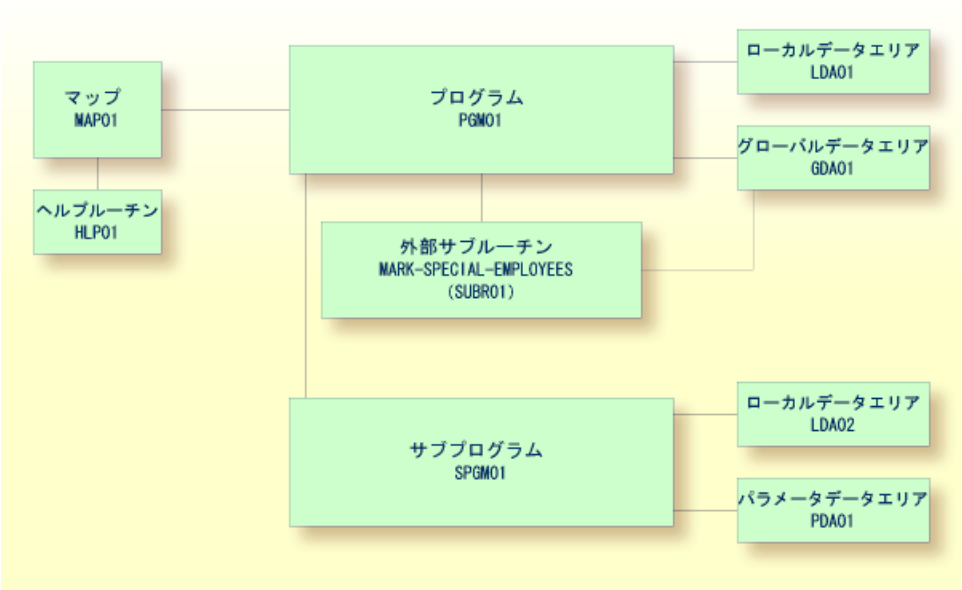
次の演習「サブプログラム」に進みます。

13 サブプログラム

サブプログラムを起動する CALLNAT ステートメントを含めるようにプログラムを拡張します。 サブプログラムでは、メインプログラムで識別された従業員が、デモデータベースの一部でもある VEHICLES ファイルに対する FIND 要求のベースとなります。 その結果、メインプログラムからの従業員情報だけでなくサブプログラムからの自動車情報も出力に含まれます。

新しいサブプログラムでは、追加のローカルデータエリアおよびパラメータデータエリアを作成する必要があります。

以下の演習を完了すると、サンプルのアプリケーションは次のモジュールで構成されます。



このchapterには次の演習が含まれています。

ローカルデータエリアの変更

前に作成したローカルデータエリアに他のフィールドを追加します。 これらのフィールドは、後で作成するサブプログラムで使用します。

手順 13.1. ローカルデータエリアに他のフィールドを追加するには

1 ローカルデータエリアに戻ります。

2 #NAME-END を含む行を選択します。

3 コンテキストメニューから、［挿入］>［データフィールド］を選択します。

Or:

次のツールバーボタンを選択します。

［データフィールドの定義］ダイアログボックスが表示されます。

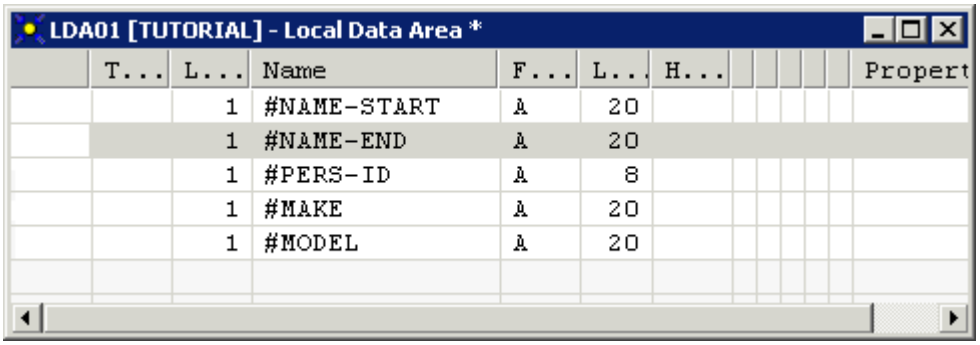
4 次のフィールドを定義します。

レベル	名前	フォーマット	長さ
1	#PERS-ID	A	8
1	#MAKE	A	20
1	#MODEL	A	20

フィールドを定義した後に、［追加］ボタンを選択します。

5 すべてのフィールドを定義した後に、［終了］ボタンを選択します。

ローカルデータエリアが次のように表示されます。



6 ローカルデータエリアを格納します。

既存のローカルデータエリアからのパラメータデータエリアの作成

パラメータデータエリア（PDA）は、Naturalプログラムおよび後で作成するサブプログラム間で受け渡すデータパラメータを指定するために使用します。 パラメータデータエリアは、サブプログラムで参照されます。

ローカルデータエリアにわずかな変更を加えるだけで、パラメータデータエリアを作成できます。つまり、ローカルデータエリアの2つのデータフィールドを削除してから、変更したデータエリアをパラメータデータエリアとして保存します。 元のローカルデータエリアはそのまま残ります。

Note: また、［オブジェクト］メニューの［新規作成］>［パラメータデータエリア］を選択して、パラメータデータエリアを作成することもできます。

手順 13.2. パラメータデータエリアを作成するには

1 ローカルデータエリアで、フィールド #NAME-START および #NAME-END を削除します。

2 ［オブジェクト］メニューの［名前を付けて保存］を選択します。

［名前をつけて保存］ダイアログボックスが表示されます。 パラメータデータエリアの名前として「PDA01」と指定します。

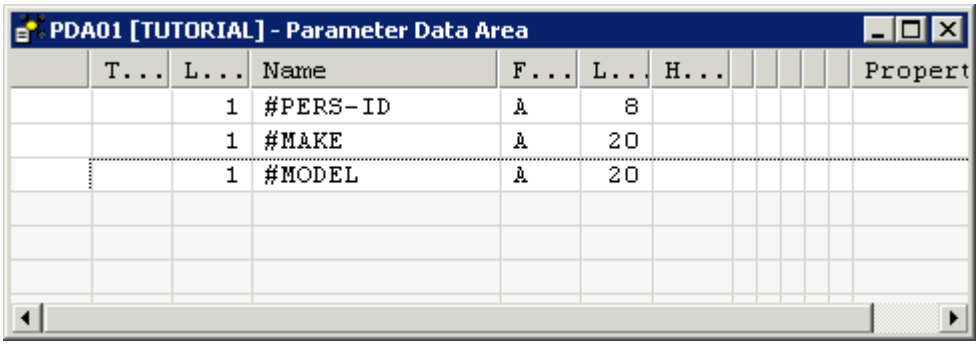
3 プログラムも含むライブラリ（**TUTORIAL** ノード）が選択されていることを確認します。

4 ［タイプ］ドロップダウンリストボックスの［パラメータ］を選択します。

5 ［**OK**］ボタンを選択します。

新しい名前とタイプがエディタウィンドウのタイトルバーに表示されます。ライブラリワークスペースで、新しいパラメータデータエリアが **Parameter Data Areas** ノードに表示されます。

パラメータデータエリアが次のように表示されます。



6 パラメータデータエリアを格納します。

7 パラメータデータエリアのエディタウィンドウを閉じます。

異なるビューを含む別のローカルデータエリアの作成

2 つ目のローカルデータエリアを作成し、VEHICLES データベースファイルの DDM からフィールドをインポートします。 この DDM もシステムライブラリ SYSEXDDM に用意されています。

このローカルデータエリアは、サブプログラムで参照されます。

手順 13.3. ローカルデータエリアを作成するには

1 ライブラリワークスペースで、プログラムも含まれているライブラリ（**TUTORIAL** ノード）を選択します。

2 コンテキストメニューから、［新規作成］>［ローカルデータエリア］を選択します。

Or:

次のツールバーボタンを選択します。

エディタウィンドウが表示されます。

3 ［挿入］メニューの［インポート］を選択します。

［データフィールドのインポート］ダイアログボックスが表示されます。

4 ［ライブラリ］ドロップダウンリストボックスから［**SYSEXDDM**］を選択します。

［**DDM**］オプションボタンを選択します。

［オブジェクト］リストボックスで、**VEHICLES** という名前のサンプル DDM を選択します。

5 Ctrl キーを押して、［インポータブルデータ］リストボックスで次のフィールドを選択します。

```
PERSONNEL-ID
CAR-DETAILS
MAKE
MODEL
```

6 ［インポート］ボタンを選択します。

［ビューの定義］ダイアログボックスが表示されます。

7 ビューの名前として「VEHICLES-VIEW」と指定します。

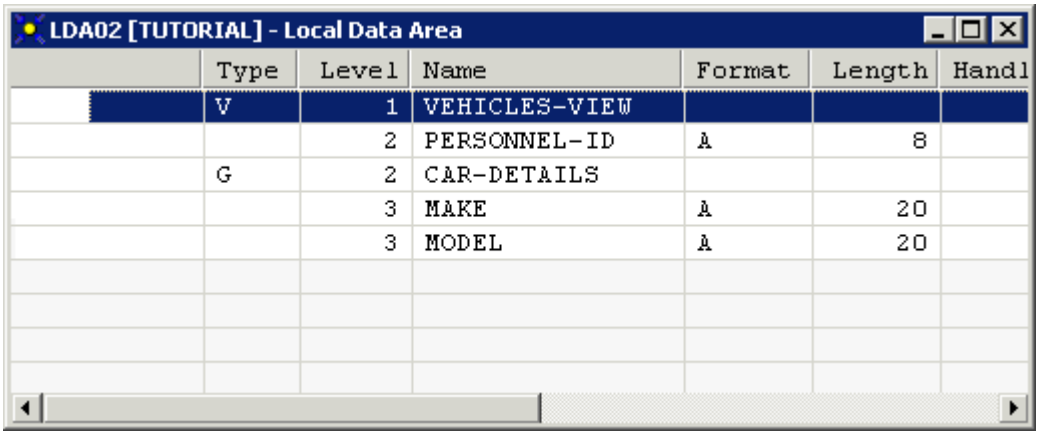
8 ［**OK**］ボタンを選択します。

9 ［終了］ボタンを選択して、［データフィールドのインポート］ダイアログボックスを閉じます。

10 新しいローカルデータエリアを格納します。

11 ローカルデータエリアの名前を指定するよう指示されたら、「LDA02」と入力し、［**OK**］ボタンを選択します。

ローカルデータエリアが次のように表示されます。



12 ローカルデータエリアのエディタウィンドウを閉じます。

サブプログラムの作成

パラメータデータエリアおよびローカルデータエリアを使用して VEHICLES ファイルから情報を取得するサブプログラムを作成します。 サブプログラムではプログラム PGM01 から渡された職員 ID を受け取り、VEHICLES ファイルの検索ベースとしてこの ID を使用します。

手順 13.4. サブプログラムを作成するには

1 ライブラリワークスペースで、プログラムも含まれているライブラリ（**TUTORIAL** ノード）を選択します。

2 コンテキストメニューから、［新規作成］>［サブプログラム］を選択します。

エディタウィンドウが表示されます。

3 次のように入力します。

```
DEFINE DATA
  PARAMETER USING PDA01
  LOCAL USING LDA02
END-DEFINE
*
FD1. FIND (1) VEHICLES-VIEW
  WITH PERSONNEL-ID = #PERS-ID
  MOVE MAKE (FD1.) TO #MAKE
  MOVE MODEL (FD1.) TO #MODEL
  ESCAPE BOTTOM
END-FIND
*
END
```

このサブプログラムから、従業員の社用車の車種およびモデルが特定の職員 ID に対して返されます。

FIND ステートメントは、検索条件 #PERS-ID に基づいて、データベースから一連のレコード（ここでは 1 レコード）を選択します。

フィールド #PERS-ID には、プログラム PGM01 から渡された PERSONNEL-ID の値が入ります。サブプログラムでは、VEHICLES ファイルの検索ベースとしてこの値が使用されます。

4 サブプログラムを格納します。

5 サブプログラムの名前を指定するよう指示されたら、「SPGM01」と入力し、［**OK**］ボタンを選択します。

6 サブプログラムのエディタウィンドウを閉じます。

プログラムからのサブプログラムの参照

サブプログラムは、CALLNAT ステートメントを使用してメインプログラムから呼び出されます。サブプログラムは、CALLNAT ステートメントによってのみ呼び出すことができます。単独で実行することはできません。 サブプログラムには、呼び出し側オブジェクトが使用するグローバルデータエリアへのアクセスがありません。

メインプログラムから指定のサブプログラムへは、サブプログラムの DEFINE DATA PARAMETER ステートメントで参照される一連のパラメータを介してデータが渡されます。

サブプログラムのパラメータデータエリアで定義される変数は、CALLNAT ステートメントの変数と同じ名前でなくてもかまいません。 パラメータはアドレスによって渡されるため、順序、フォーマット、および長さが一致していれば十分です。

定義したサブプログラムが使用されるように、メインプログラムを変更します。

手順 13.5. メインプログラムでサブプログラムを使用するには

1 プログラムエディタに戻ります。

2 DISPLAY ステートメントの直前に、次のように挿入します。

```
RESET #MAKE #MODEL
CALLNAT 'SPGM01' PERSONNEL-ID #MAKE #MODEL
```

RESET ステートメントにより、#MAKE および #MODEL の値が空値に設定されます。

3 DISPLAY ステートメントを含む行を削除し、次のように置き換えます。

```
WRITE TITLE
      / '*** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***'
      / '***     ARE MARKED WITH AN ASTERISK        ***'//
*
DISPLAY   1X '//N A M E' NAME
          1X '//DEPT'    DEPT
          1X '/LV/DUE'   LEAVE-DUE
             ' '         #MARK
          1X '//MAKE'    #MAKE
          1X '//MODEL'   #MODEL
```

WRITE TITLE ステートメントで定義されたテキストが、各出力ページの先頭に表示されます。WRITE TITLE ステートメントは、デフォルトのページタイトルを無効にします。各ページの先頭にこれまで表示されていた情報（ページ番号、日時）は表示されなくなります。各スラッシュ（/）により、後続の情報が新しい行に表示されます。

サブプログラムから追加の自動車情報が返されるため、出力列のサイズを変更する必要があります。 ヘッダーは短くなります。 アスタリスクを表示する列（#MARK）には、ヘッダーは表示されません。列間には1文字分の空白が挿入されます（1X）。ヘッダー内の各スラッシュにより、後続の情報が同じ列の新しい行に表示されます。

プログラムは次のようになります。

```
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
```

```
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    IF LEAVE-DUE >= 20 THEN
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
    RESET #MAKE #MODEL
    CALLNAT 'SPGM01' PERSONNEL-ID #MAKE #MODEL
*
    WRITE TITLE
      / '*** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***'
      / '***     ARE MARKED WITH AN ASTERISK        ***'//
*
    DISPLAY   1X '//N A M E' NAME
              1X '//DEPT'    DEPT
              1X '/LV/DUE'   LEAVE-DUE
                 ' '         #MARK
              1X '//MAKE'    #MAKE
              1X '//MODEL'   #MODEL
*
  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT
*
END
```

4 プログラムを実行します。

5 開始名として「JONES」と入力し、Enter キーを押します。

結果のリストは次のようになります。

```
              *** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***
              ***     ARE MARKED WITH AN ASTERISK        ***


                           LV
      N A M E         DEPT  DUE           MAKE                MODEL
------------------- ------ --- - -------------------- --------------------

JONES                SALE30  25 * CHRYSLER             IMPERIAL
JONES                MGMT10  34 * CHRYSLER             PLYMOUTH
JONES                TECH10  11   GENERAL MOTORS       CHEVROLET
JONES                MGMT10  18   FORD                 ESCORT
JONES                TECH10  21 * GENERAL MOTORS       BUICK
JONES                SALE00  30 * GENERAL MOTORS       PONTIAC
JONES                SALE20  14   GENERAL MOTORS       OLDSMOBILE
JONES                COMP12  26 * DATSUN               SUNNY
JONES                TECH02  25 * FORD                 ESCORT 1.3
```

6 Esc キーを押して、出力ウィンドウを閉じます。

7 プログラムを格納します。

これで、チュートリアルがすべて完了しました。

索引

ら

る

れ

ろ