Natural

First Steps

Version 6.3.13 for UNIX

October 2012

Natural

This document applies to Natural Version 6.3.13 for UNIX.

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

Copyright © 1992-2012 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://documentation.softwareag.com/legal/.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

Document ID: NATUX-NNATFIRSTSTEPS-6313-20121005

Table of Contents

Preface v
1 About this Tutorial 1
Prerequisites 2
About the Sample Application 2
2 Getting Started with Natural 5
Invoking Natural's Main Menu 6
Libraries 7
Issuing Commands 7
Creating a User Library 7
Programming Modes 9
3 Hello World! 11
Creating a Program 12
Running a Program 13
Correcting Program Errors 14
Stowing a Program 15
Displaying Information about a Program 16
Displaying the Content of the Current Library 17
Setting the Editor Profile Options 19
4 Database Access 23
Saving Your Program Under a New Name 24
Defining the Required Data Using a View 25
Reading Data from a Database 28
Reading Selected Data from a Database 30
5 User Input 33
Allowing for User Input 34
Designing a Map for User Input 36
Invoking the Map from Your Program 47
Ensuring that an Ending Name is Always Used 49
6 Loops and Labels 51
Allowing Repeated Usage 52
Displaying a Message Indicating that Information was not Found 54
7 Inline Subroutines 57
Defining the Inline Subroutine 58
Performing the Inline Subroutine 59
8 Processing Rules and Helproutines 63
Defining a Processing Rule 64
Defining a Helproutine 67
9 Local Data Areas 71
Creating a Local Data Area 72
Defining Data Fields 73
Importing the Required Data Fields from a DDM 75
Referencing the Local Data Area from Your Program 78
10 Global Data Areas 81

Creating a Global Data Area from an Existing Local Data Area 82
Adapting the Local Data Area 84
Referencing the Global Data Area from Your Program 85
11 External Subroutines 87
Creating an External Subroutine 88
Referencing the External Subroutine from Your Program 89
12 Subprograms 93
Modifying the Local Data Area 94
Creating a Parameter Data Area from an Existing Local Data Area 96
Creating Another Local Data Area Containing a Different View 97
Creating a Subprogram 99
Referencing the Subprogram from Your Program 100

Preface

This tutorial provides a very simple and brief introduction to programming with Natural and to using the Natural editors.

Important: It is important that you read the following topics in the sequence indicated below, and that you work through all exercises in these topics in the same sequence as they appear in this tutorial. Problems may occur if you skip an exercise.

About this Tutorial	Prerequisites and what you will learn in the course of this tutorial.
Getting Started with Natural	How to invoke Natural's main menu. How to create the library that will be used in this tutorial. Information on Natural's programming modes and the mode that is required for this tutorial.
Hello World!	How to create, run and stow your first short program. How to display the content of the current library. Information on some options which control your editor profile.
Database Access	How to read specific data from a database and display the output.
User Input	How to prompt the user for information and how to design a map for user input. How to ensure that a specific value is always used (here: an ending name), even if it has not been specified by the user.
Loops and Labels	How to define a repeat loop and labels for different loops. How to display a message when specific information (here: the starting name entered by the user) was not found.
Inline Subroutines	How to define and invoke an inline subroutine (that is: a subroutine which is coded directly in the program).
Processing Rules and Helproutines	How to define a processing rule (here: a message that is to appear when the user does not specify a starting name) and a helproutine (here: a help text for the field in which the user has to enter a starting name).
Local Data Areas	How to relocate the field definitions from the program to a local data area outside the program.
Global Data Areas	How to define a global data area which can be shared by multiple programs or routines.
External Subroutines	How to define and invoke an external subroutine (that is: a subroutine which is stored as a separate object outside the program).
Subprograms	How to define a parameter data area for a subprogram. How to define and invoke a subprogram.

1 About this Tutorial

- Prerequisites 2
- About the Sample Application 2

As a first-time user, you are recommended to work through this tutorial to obtain a basic understanding of specific features of the Natural programming environment.

The layout of the example screens provided in the tutorial and the behavior of Natural described here can differ from your results. For example, the command or message line may appear in a different screen position, or the execution of a Natural command may be protected by security control. The default settings in your environment depend on the system parameters set by your Natural administrator.

Prerequisites

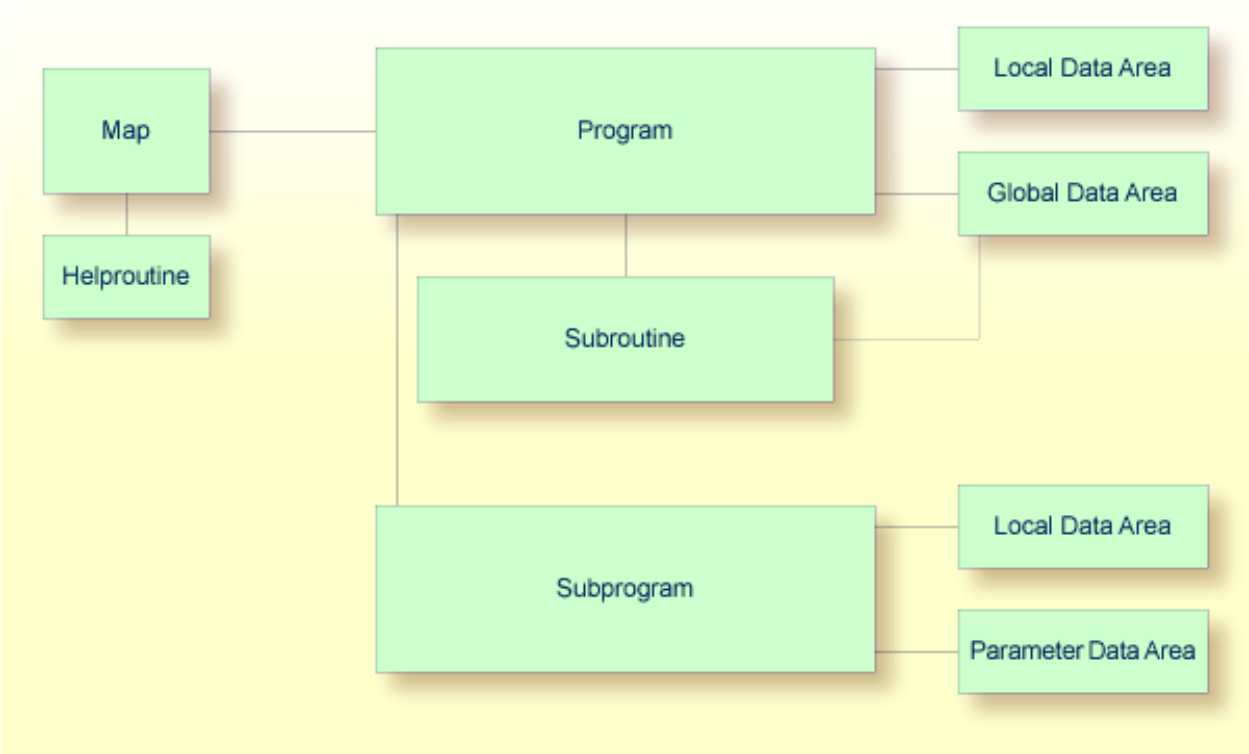
To perform all steps of this tutorial, the demo database `SAG-DEMO-DB` must be active. If it is not active, ask your administrator to start it.

The system library `SYSEXDDM` which contains the sample DDMs that are used in this tutorial (`EMPLOYEES` and `VEHICLES`) must have been defined as a steplib. A steplib is a library in which Natural searches if an object is not found in the current library. If `SYSEXDDM` has not been defined as a steplib, an error occurs when you try to define the views for the sample DDMs; contact your administrator in this case.

About the Sample Application

This tutorial illustrates how an application can be structured as a group of modules. It is not intended to provide an example of how an application should be built.

After you have written your first short Hello World program, you will write a program which reads employees information from a database and displays the output. The user will be prompted to enter a starting name and ending name for the output. You will enhance your program step by step by moving specific parts of your program to external modules. When you have completed all exercises of this tutorial, your application will be structured as follows:



Note: This tutorial describes how to create a map which is normally used in a character-oriented environment (such as a mainframe). For a graphical user interface, you would create a dialog. However, this is not part of this tutorial.

You can now proceed with your first exercise: *Getting Started with Natural*.

2 Getting Started with Natural

- Invoking Natural's Main Menu 6
- Libraries 7
- Issuing Commands 7
- Creating a User Library 7
- Programming Modes 9

Invoking Natural's Main Menu

The way you invoke Natural and its main menu depends on how the system has been configured at your site. For most installations, you invoke Natural as described below.

To invoke Natural's main menu

- Enter the following command at the UNIX system prompt:

```
natural
```

The main menu appears.

```
 2009-06-30                        NATURAL                 Library: SYSTEM
 09:22:05                  V 6.3.7 Software AG 2009        Mode   : REPORT
 User: SAG                                               Work Area : empty
+-----------------------------------------------------------------------------+
¦Library          Direct           Services         OS                Fin      ¦
+-----------------------------------------------------------------------------+
 Select Library
```

Libraries

All Natural objects required for creating an application are stored in Natural libraries in Natural system files. There is a system file for system programs (FNAT) and a system file for user-written programs (FUSER).

Natural thus distinguishes system libraries and user libraries. The system libraries, which start with the letters "SYS", are reserved for Software AG purposes only. A user library contains all user-defined objects (for example, programs and maps) which make up an application. The name of a user library must not start with the letters "SYS".

The field **Library** in the top right-hand corner of the Natural main menu shows the name of the library where you are currently logged on.

Issuing Commands

In Natural, you can perform a function either by selecting it from a sequence of menus and selection windows, or by entering a Natural system command directly.

To select a menu, use the arrow keys. When the required menu is highlighted, press ENTER. It is also possible to enter the first character of a menu name; you need not press ENTER in this case.

When the resulting window contains a list of options, use the arrow keys to select the required option and press ENTER. In some windows, it is also possible to enter the first character of the function; you need not press ENTER in this case.

If you want to close a window without further action, press ESC.

The input of a Natural command is not case-sensitive. After you have entered a Natural command, you choose the ENTER key. ENTER confirms the action and executes the command or invokes an extra confirmation window where you explicitly acknowledge command execution.

Creating a User Library

You will now create a user library with the name TUTORIAL. This library is to contain all Natural objects that you will create in the course of this tutorial.

A library can be created in different ways. The following instructions illustrate how to use the **Direct** menu in which you enter Natural system commands, and how to use the **Library** menu to achieve the same result.

To create a user library using the Direct menu

- Select the **Direct** menu and press ENTER. In the resulting **Direct Command** window enter the following and press ENTER:

```
LOGON TUTORIAL
```

where "TUTORIAL" is the name of the library that you create.

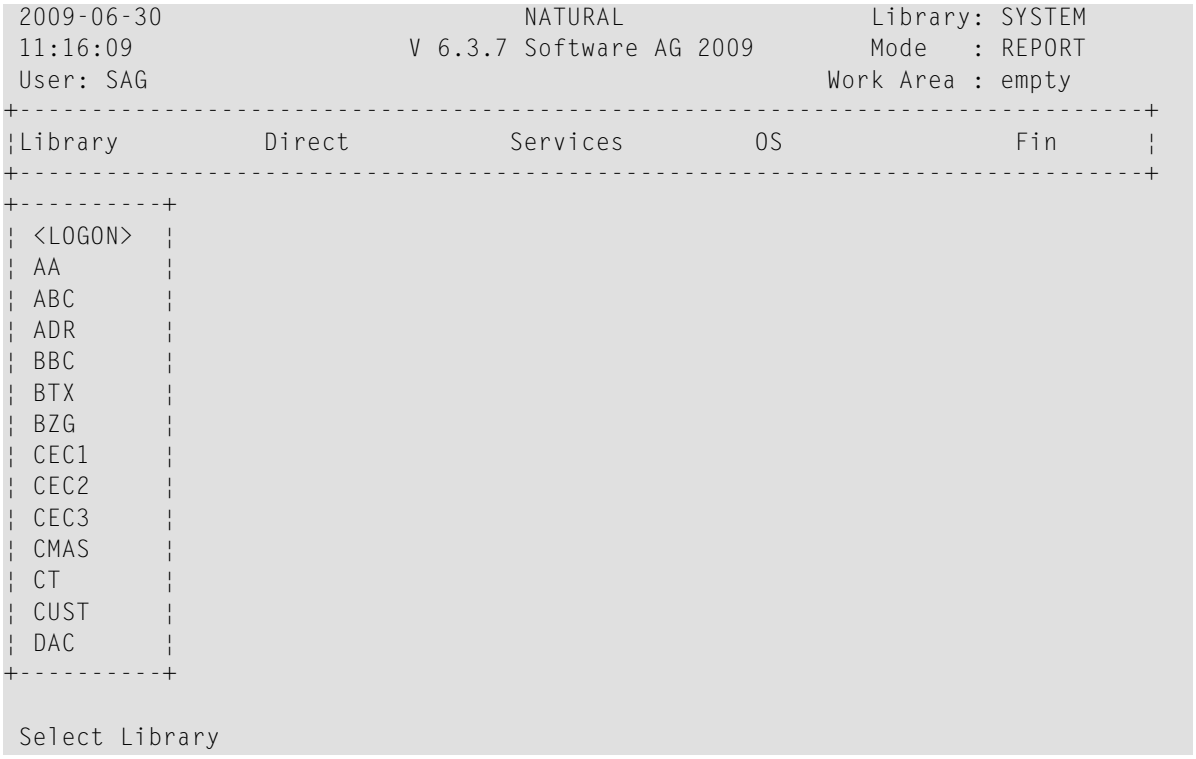
LOGON is a system command which is used for two purposes:

- to log on to an existing library, or
- to create a new library when a library with the specified name does not exist.

To create a user library using the Library menu

1 Select the **Library** menu and press ENTER.

A window appears providing access to all existing libraries.



2 Use the arrow keys to select the first entry which is **<LOGON>** and press ENTER.

The following window appears. You can either log on on to an existing library or add a new library.

```
+-- New Library: --+
¦                  ¦
+------------------+
```

3 Specify the name "TUTORIAL".

When the name of the library has eight characters, you need not press ENTER. In this case, you are automatically logged on to the library after you have entered the last character of the name.

Since this is a new library which does not yet contain any objects, a message is shown indicating that the library is currently emtpy.

Notes:

1. If you want to log on to an existing library, you can also use the arrow keys to scroll through all libraries in the window. Or you can enter a character to proceed to the libraries that start with this character. Press ENTER to log on to the highlighted library.
2. If you log on to a library which already contains objects, a list of objects in this library is shown. This list is is not shown when you log on using the **Direct Command** window as explained in the previous exercise.

Programming Modes

The programming mode is indicated in the **Mode** field at the top right-hand corner of the menu.

Natural provides two different programming modes:

- **Structured Mode**
  Structured mode is intended for the implementation of complex applications with a clear and well-defined program structure. It is recommended to use structured mode exclusively.
- **Reporting Mode**
  Reporting mode is only useful for the creation of adhoc reports and small programs which do not involve complex data and/or programming constructs.

Important: This tutorial requires that structured mode is active. If you try to run your program in reporting mode, `END-IF`, `END-READ` and `END-REPEAT` will cause errors.

If reporting mode is currently active, proceed as described below.

To switch from reporting mode to structured mode

- Invoke the **Direct Command** window (by choosing the **Direct** menu), enter the following system command and press ENTER:

```
GLOBALS SM=ON
```

You can now proceed with your first program: ***Hello World!***

3 Hello World!

- Creating a Program 12
- Running a Program 13
- Correcting Program Errors 14
- Stowing a Program 15
- Displaying Information about a Program 16
- Displaying the Content of the Current Library 17
- Setting the Editor Profile Options 19

Creating a Program

You will now write your first short program which displays "Hello World!". It will be stored in the library you have created previously.

To create a new program

1 Make sure that you have logged on to the library named `TUTORIAL`.

2 Invoke the **Direct Command** window and enter one of the following:

```
EDIT PROGRAM
```

```
E P
```

System commands may be abbreviated. `E P` is the abbreviated form of `EDIT PROGRAM`.

The program editor appears. It is currently empty.

```
>> -----------Columns 001 072 <<  Program          Lines         User SAG
Command ===>                                       Mode   Struct Lib  TUTORIAL
****** ***************************** top of data ******************************
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
****** **************************** bottom of data ****************************
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Save  Exit  Run   Rfind Stow  -     +     Check Home  Undo  Canc
```

Caution: Do not press ENTER now. When you press ENTER, you leave insert mode. To enter code, you then have to return to insert mode as follows: enter the line command `I` at the beginning of the line which contains the words "top of data" and press ENTER.

3 Enter the following code in the program editor:

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

When you press ENTER twice after the last line of code has been added, insert mode is switched off.

Comment lines start with an asterisk (*) followed by at least one blank or a second asterisk. When you forget to enter the blank or second asterisk, Natural assumes that you have specified a system variable; this will result in an error.

If you want to insert empty lines in your program, you should define them as comment lines. This is helpful, if you want to access your program from different platforms (Windows, mainframe, UNIX or OpenVMS). With the mainframe version of Natural, for example, the default is that empty lines are automatically deleted when you press ENTER.

You can also insert comments at the end of a statement line. In this case, the comment starts with a slash followed by an asterisk (/*).

The text that is to be shown in the output is defined with the `DISPLAY` statement. It is enclosed in quotation marks.

The `END` statement is used to mark the physical end of a Natural program. Each program must end with `END`.

When you press ENTER, it may happen that all of your lower-case characters are translated to upper-case characters. This behavior is defined in the editor profile (which is explained later).

Running a Program

The system command `RUN` automatically invokes the system command `CHECK` which checks the program code for errors. If no error is found, the program is compiled on the fly and then executed.

Notes:

1. `CHECK` is also available as a separate command.
2. Natural also provides the system command `EXECUTE` which uses the stowed version of your program (stowing a program is explained later in this tutorial). In contrast to this, the `RUN` command always uses your latest modifications to the program.

To run a program

1 In the program editor's command line, enter one of the following:

```
RUN
```

```
R
```

Note: In the program editor, you can press PF10 (Home) to place the cursor in the command line.

When your code is syntactically correct, the output contains the text you have defined.

```
MORE
Page      1                                                  10-11-22  12:07:25


Hello world!
```

2 Press ENTER to return to the program editor.

Correcting Program Errors

You will now create an error in your Hello World program and then run the program once more.

To correct an error

1 Delete the second quotation mark in the line containing the `DISPLAY` statement.

2 Run the program once more as described above.

When the error is found, an error message is displayed.

```
>> -----------Columns 001 072 <<  Program          Lines  4      User SAG
Command ===> run                                   Mode   Struct Lib  TUTORIAL
****** ***************************** top of data ******************************
000010 * The "Hello world!" example in Natural.
000020 *
000030 DISPLAY "Hello world!
000040 END /* End of program
+-------------------------- Error in File:          --------------------------+
¦NAT0305 Text string must begin and end on the same line.                     ¦
¦             VVVVVVVVVVVVVV                                                  ¦
¦0030 DISPLAY "Hello world!                                                   ¦
+-------------------------------------------------------------------------------+









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Save  Exit  Run   Rfind Stow  -     +     Check Home  Undo  Canc
```

3 Correct the error in the window which displays the error message, that is: insert the missing quotation mark at the end of the line.

4 Press ENTER to find the next error.

In this case, no more errors are found and the output is shown.

5 Press ENTER to return to the program editor.

Stowing a Program

When you stow a program, it is compiled and both source code and a generated program are stored in the Natural system file.

Like the `RUN` command, the system command `STOW` automatically invokes the `CHECK` command. A program is only stowed when it is syntactically correct.

Note: If you want to save the changes to your program, even if the program contains a syntactical error (for example, if you want to suspend your work until the next day), you can use the system command `SAVE`. When a program is saved for the first time, you also have to specify a name. For example: `SAVE HELLO`.

To stow a program

- In the program editor's command line, enter the following:

```
STOW HELLO
```

  where "HELLO" is the name with which your program is to be stored.

  Note: When a program has already been given a name, it is sufficient to enter `STOW` in the command line (without a program name) or to press PF6.

Displaying Information about a Program

The `LIST` command is useful to find out whether only the source code or both source code and a generated program are available for an object.

To display information about a program

1 In the program editor's command line, enter one of the following:

```
LIST DIR HELLO
```

```
L DIR HELLO
```

  The following screen appears. The information provided with **Cataloged on** is only available when the object has been stowed.

```
>> -----------Columns 001 072 <<  Program HELLO    Lines  4      User SAG
Command ===> list dir hello                        Mode   Struct Lib  TUTORIAL
****** ****************************** top of data ****************************
+---------------------------- List Directory HELLO -----------------------------+
¦ Directory of Program HELLO                  Saved on ... 2009-06-30 16:37:00 ¦
¦ ------------------------------------------------------------------------------ ¦
¦ Library .... TUTORIAL   User-ID ...... SAG       Mode .. Structured           ¦
¦ OP-System .. SUN_SOLA                                                         ¦
¦ NAT-Ver .... V 6.3.7                                                          ¦
¦ Size ............................... 108 Bytes                                ¦
¦                                                                               ¦
¦ Directory of Program HELLO                  Cataloged on 2009-06-30 16:37:00 ¦
¦ ------------------------------------------------------------------------------ ¦
¦ Library .... TUTORIAL   User-ID ...... SAG       Mode .. Structured           ¦
¦ OP-System .. SunOS                          OP-Version ..5.8Generic_10852     ¦
¦ NAT-Ver .... V 6.3.7                                                          ¦
¦ Size ............................... 330 Bytes                                ¦
¦ Endian mode: Big                                                              ¦
¦                                                          ENTER to continue    ¦
+---------------------------------------------------------------------------------+

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Save  Exit  Run   Rfind Stow  -     +     Check Home  Undo  Canc
```

2 Press ENTER to return to the program editor.

Displaying the Content of the Current Library

The `LIST` command can also be used to display a list of all Natural objects in the current library. This is helpful, for example, if you decide at some point during this tutorial that you want to delete one or more of your Natural objects in order to start again from the very beginning.

To display a list of Natural objects

1 In the program editor's command line, enter one of the following:

```
LIST *
```

```
L *
```

The following window appears. It lists the program you have just created.

```
>> -----------Columns 001 072 <<  Program HELLO    Lines  4      User SAG
Command ===> l *                                    Mode   Struct Lib  TUTORIAL
****** ***************************** top of data ******************************
000010 * The "Hello world!" example in Natural.
000020 *
000030 DISPLAY "Hello world!"
0+---------------------------------- List * * ----------------------------------+
*¦ Cmd Name     Type        SM S/C Userid   SRC Date         GP Date           ¦
 ¦ --- -------- ----------- -- --- -------- ---------------- ---------------- ¦
 ¦ <DIRECT COMMAND>                                                           ¦
 ¦     HELLO    Program     S  S/C SAG      16:37 2009-06-30 16:37 2009-06-30 ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
 ¦                                                                            ¦
E¦                                                                            ¦
 +----------------------------------------------------------------------------+
```

2 To find out which commands are available, enter a question mark (?) in the **Cmd** column next to your program.

The following window appears.

```
+--------------------------+
¦ C Check                  ¦
¦ D Read                   ¦
¦ E Edit                   ¦
¦ L List                   ¦
¦ I List Dir               ¦
¦ H Hardcopy               ¦
¦ R Run                    ¦
¦ X Execute                ¦
¦ S Stow                   ¦
¦ U Scratch                ¦
¦ . End                    ¦
+--------------------------+
```

Note: **Scratch** is used to delete an object.

3 Do not apply any changes right now. Press ESC to close the window without selecting any command.

4 Press ESC once more to return to the program editor.

Setting the Editor Profile Options

When working with the Natural program editor, an editor profile can be defined per user. This tutorial uses the default settings of the editor profile named SYSTEM. Some important settings are mentioned below.

To check the editor profile options

1 In the program editor's command line, enter the following:

```
PROFILE
```

The main menu of the editor profile appears.

```
13:57:44         **** Program Editor Profile ****                    09.06.30
                            Main Menu

  Profile Name ... SYSTEM


       _  Save

       _  Modify

       _  Read

       _  Technical Info









Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  Modi  Save  Read  Tech                          Canc
```

When a user-specific editor profile does not exist, the default profile SYSTEM is displayed. This default profile can be used to create a user-specific profile. When a user-specific profile exists already, it is displayed instead of the SYSTEM profile.

2 Mark the option **Modify** and press ENTER.

Or:

Press PF4.

The following screen appears.

```
14:05:36          **** Program Editor Profile ****                  09.06.30
                         Modify Defaults

  User ID ........ SYSTEM


       _  PA/PF-Keys

       _  Commands

       _  Find

       _  General









Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Keys  Exit        Ctrl        Find  Genl                    Canc
```

3 Mark the option **Commands** and press ENTER.

The following screen appears.

```
14:04:39            **** Program Editor Profile ****                09.06.30
                         Modify Editor Defaults

User ID ....... SYSTEM


aorder ............. OFF           hex ................ OFF
autosave ........... OFF           justify ............ LEFT
caps ............... OFF           limit .............. OFF
cols ............... OFF           log ................ OFF
decimal character ... .            mask line .......... OFF
empty .............. OFF           message line ....... ON
escape ............. OFF           mso ................ ON
escape character .... .            scroll mode ........ PAGE
fix ................ OFF           tabs ............... OFF
fixlen ............. 48            tabulator character . ^




Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                 Canc
```

Check the setting of the following options:

- **caps**
  Specifies whether data are to be translated into upper case.
- **empty**
  Specifies whether lines containing only space characters are to be deleted automatically.
- **escape**
  Specifies whether the escape character is to be used to precede line commands.

This tutorial assumes that the above options are set to "OFF".

4 If "ON" is currently defined for any of the above options, overwrite it with "OFF".

5 Press PF3 or enter `EXIT` in the command line.

6 Press PF3 repeatedly until the main menu of the editor profile is shown again.

7 When a user-specific profile has not yet been created, overwrite the profile name `SYSTEM` with your user ID.

When a user-specific profile exists already, proceed with the next step.

8 Mark the option **Save** and press ENTER.

Or:

Press PF5.

9 Press PF3.

Or:

Enter `EXIT` in the command line.

The program editor is shown again. Any new settings will now be used in the program editor.

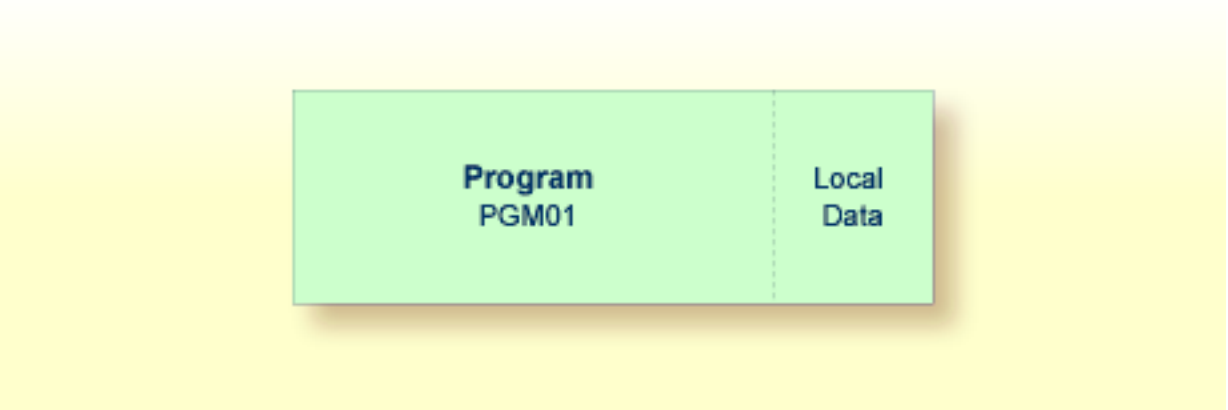
You can now proceed with the next exercises: ***Database Access***.

4 Database Access

- Saving Your Program Under a New Name 24
- Defining the Required Data Using a View 25
- Reading Data from a Database 28
- Reading Selected Data from a Database 30

You will now write a short program which reads specific data from a database file and displays the corresponding output.

When you have completed the exercises below, your sample application will consist of just one module (the data fields that are used by the program are defined within the program):



Saving Your Program Under a New Name

You will now create a new program which will be used in the remainder of this tutorial. It will be created by saving your Hello World program under a new name.

To save the program under a new name

1 In the program editor's command line, enter one of the following:

```
SAVE PGM01
```

```
SA PGM01
```

The current program is saved with the new name `PGM01`. The program named `HELLO` is still shown in the program editor.

2 Read the newly created program into the program editor by entering the following in the program editor's command line:

```
READ PGM01
```

The program name which is displayed in the program editor changes to `PGM01`.

3 Delete all code in the program editor. To do so, enter the following line command at the beginning of each line to be deleted and press ENTER:

```
D
```

Example:

```
>> -----------Columns 001 072 <<  Program PGM01    Lines  4      User SAG
Command ===>                                        Mode   Struct Lib  TUTORIAL
****** ***************************** top of data ******************************
D00010 * The "Hello world!" example in Natural.
D00020 *
D00030 DISPLAY "Hello world!"
D00040 END /* End of program
****** **************************** bottom of data ****************************
```

Or:

Enter the following line command at the beginning of the first line and press ENTER:

```
D4
```

where the number after the command indicates the number of lines to be deleted.

Or:

Enter the following line command in the command line, move the cursor to the first line to be deleted and press ENTER:

```
:D4
```

Line commands can also be entered in the command line of the editor screen. In this case, the command must be preceded by a colon (:). It always applies to the line marked by the cursor.

Defining the Required Data Using a View

The database file and the fields that are to be used by your program have to be specified between `DEFINE DATA` and `END-DEFINE` at the top of the program.

For Natural to be able to access a database file, a logical definition of the physical database file is required. Such a logical file definition is called a data definition module (DDM). The DDM contains information about the individual fields of the file. DDMs are usually defined by the Natural administrator.

To be able to use the database fields in a Natural program, you must specify the fields from the DDM in a view. For this tutorial, we will use the DDM for the `EMPLOYEES` database file.

Since you have deleted all lines of your previous Hello World program, the program editor currently looks as follows:

```
>> -----------Columns 001 072 <<  Program PGM01    Lines         User SAG
Command ===>                                       Mode   Struct Lib  TUTORIAL
****** ***************************** top of data ******************************
****** **************************** bottom of data ****************************
```

Before you can enter the code for your program, you have to insert blank lines.

To insert blank lines

- Enter the following line command at the beginning of the line which contains the words “top of data” and press ENTER:

  ```
  I
  ```

  This inserts one blank line and the editor switches to insert mode. You can now enter your program code (see below). When you press ENTER, a new line is inserted. If you do not enter any data in a newly inserted line (which has a number of apostrophes instead of a line number) and press ENTER, the editor leaves insert mode and the blank line is deleted.

  Tip: To insert a blank line below `LOCAL` (as indicated below), enter an asterisk (*) in this line. When insert mode is no longer active (after you have entered the last line of the program and have pressed ENTER twice), you can remove the asterisk. The resulting blank line will not be deleted in this case.

To specify the `DEFINE DATA` block

- Enter the following code in the program editor:

  ```
  DEFINE DATA
  LOCAL

  END-DEFINE
  *
  END
  ```

  `LOCAL` means that the variables that you will define with the next step are local variables which apply only to this program.

To display the data fields from the DDM in a split screen

1 In the program editor's command line, enter the following:

  ```
  SPLIT VIEW EMPLOYEES SHORT
  ```

  `SHORT` indicates that the data fields are to be listed in short form (that is, only the Adabas short names and corresponding Natural field names are displayed).

The screen is divided into two sections. The data fields from the DDM displayed in the lower half of the screen. It is not possible to edit the data in the lower half of the screen.

```
>> -----------Columns 001 072  <<  Program PGM01    Lines  6      User SAG
Command ===>                                        Mode   Struct Lib  TUTORIAL
****** ***************************** top of data ******************************
000010 DEFINE DATA
000020 LOCAL
000030
000040 END-DEFINE
000050 *
000060 END
****** **************************** bottom of data ****************************

>> -----------Columns 001 072  <<     View EMPLOYEE Lines  38     User SAG
Command ===>                                                    Lib  SYSTEM
****** ***************************** top of data ******************************
000001 DB: 020 FILE: 014  - EMPLOYEES                        DEFAULT SEQUENCE:
000002   1 AA PERSONNEL-ID                      A    8    D
000003 G 1 AB FULL-NAME
000004   2 AC FIRST-NAME                        A   20  N
000005   2 AD MIDDLE-I                          A    1  N
000006   2 AE NAME                              A   20    D
000007   1 AD MIDDLE-NAME                       A   20  N
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Save  Exit  Run   Rfind Stow  -     +     Check Home  Undo  Canc
```

2 You can now page through the view to see which data fields are used and how they have been defined. To do so, use the following commands or keys:

Command or Key	Description
SWAP	Move the cursor from the command line of the edit area to the command line of the display area (view) and vice versa. It is also possible to simply move the cursor to the required command line.
+ or PF8	Page forward in the view. The cursor must be located in the command line of the display area (view).
- or PF7	Page backward in the view. The cursor must be located in the command line of the display area (view).
SPLIT END	Terminate split-screen mode. The cursor must be located in the command line of the edit area (program).

The next step assumes that split-screen mode has been terminated.

3 Place the cursor in the first position of the line containing LOCAL and enter the following:

```
I9
```

9 blank lines are inserted.

4 Enter the following code below LOCAL:

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

5 Press ENTER.

The remaining blank lines are eliminated. However, one blank line remains with the cursor in it (the editor stays in insert mode). When you now press ENTER, you will leave insert mode.

The first line contains the name of your view and the name of the database file from which the fields have been taken. This is always defined on level 1. The level is indicated at the beginning of the line. The names of the database fields from the DDM are defined at levels 2 and 3.

Levels are used in conjunction with field grouping. Fields assigned a level number of 2 or greater are considered to be a part of the immediately preceding group which has been assigned a lower level number. The definition of a group enables reference to a series of fields (this may also be only one field) by using the group name. This is a convenient and efficient method of referencing a series of consecutive fields.

Format and length of each field is indicated in parentheses. "A" stands for alphanumeric, and "N" stands for numeric.

Reading Data from a Database

Now that you have defined the required data, you will add a READ loop. This reads the data from the database file using the defined view. With each loop, one employee is read from the database file. Name, department and remaining days of vacation for this employee are displayed. Data are read until all employees have been displayed.

Note: It may happen that an error message is displayed indicating that the transaction has been aborted. This usually happens when the non-activity time limit which is determined by Adabas has been exceeded. When such an error occurs, you should simply repeat your last action (for example, issue the RUN command once more).

To read data from a database

1 Insert the following below END-DEFINE (use the I command as described above to insert blank lines):

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

`BY NAME` indicates that the data which is read from the database is to be sorted alphabetically by name.

The `DISPLAY` statement arranges the output in column format. A column is created for each specified field and a header is placed over the column. `3X` means that 3 spaces are to be inserted between the columns.

2 Run the program.

The following output appears.

```
MORE
Page      1                                                  09-06-30  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
```

As a result of the `DISPLAY` statement, the column headers (which are taken from the DDM) are underlined and one blank line is inserted between the underlining and the data. Each column has the same width as defined in the `DEFINE DATA` block (that is: as defined in the view).

The title at the top of each page, which contains the page number, date and time, is also caused by the `DISPLAY` statement.

3 Press ENTER repeatedly to display all pages.

You will return to the program editor when all employees have been displayed.

Tip: If you want to return to the program editor before all employees have been displayed, enter `EDIT` or its abbreviation `E` at the `MORE` prompt. It is also possible to enter the terminal command `%.`, which interrupts the current Natural operation, at the `MORE` prompt. By default, each terminal command starts with the control character `%`. Your administrator, however, may have defined another control character.

Reading Selected Data from a Database

Since the previous output was very long, you will now restrict it. Only the data for a range of names is to be displayed, starting with "Adkinson" and ending with "Bennett". These names are defined in the demo database.

To restrict the output to a range of data

1 Before you can use new variables, you have to define them. Therefore, insert the following below `LOCAL`:

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

These are user-defined variables; they are not defined in demo database. The hash (#) at the beginning of the name is used to distinguish the user-defined variables from the fields defined in the demo database; however, it is not a required character.

`INIT` defines the default value for the field. The default value must be specified in pointed brackets and quotation marks.

2 Insert the following below the `READ` statement:

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

Your program should now look as follows:

```
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
```

Your program code now exceeds one screen page. To navigate in the program source, you can use the following commands or keys:

Command	Description
BOT	Go to the end of the program.
TOP	Return to the beginning of the program.
Key	**Description**
PF8	Scroll down one page in the program.
PF7	Scroll up one page in the program.

3 Run the program.

The output is shown. When you press ENTER repeatedly, you will notice that you will return to the program editor after a couple of pages (that is: when the data for the last employee named Bennett has been displayed).

4 Stow the program.

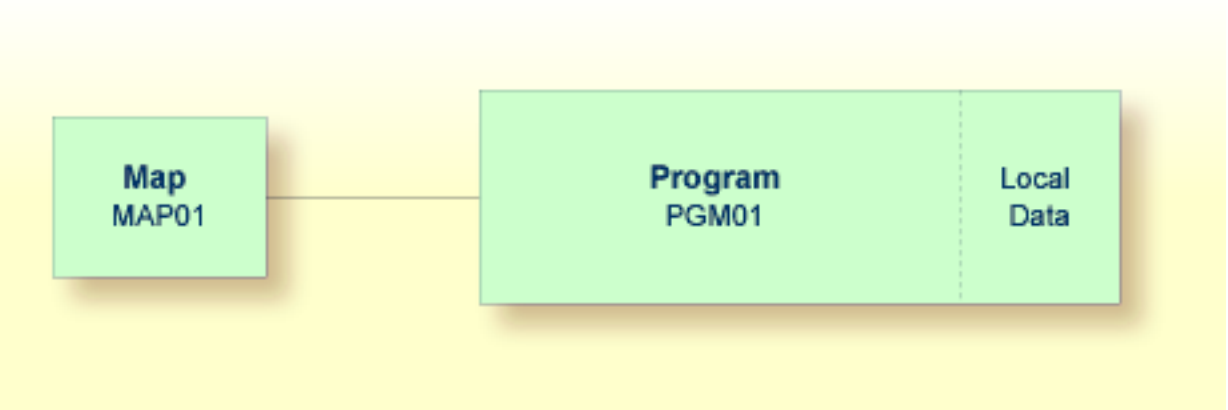
You can now proceed with the next exercises: ***User Input***.

5 User Input

- Allowing for User Input 34
- Designing a Map for User Input 36
- Invoking the Map from Your Program 47
- Ensuring that an Ending Name is Always Used 49

You will now learn how to prompt the user for data, that is: a starting name and an ending name for the output.

When you have completed the exercises below, your sample application will consist of the following modules:



Allowing for User Input

You will now modify your program so that input fields for the starting name and ending name will be shown in the output. This is done using the `INPUT` statement.

To define input fields

1 Insert the following below `END-DEFINE`:

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

The session parameter `AD` stands for “attribute definition”, its value "M" stands for “modifiable output field”, and the value "T" stands for “translate lowercase to uppercase”.

The "M" value in `AD=MT` means that the default values defined with `INIT` (that is: "ADKINSON" and "BENNETT") will be shown in the input fields. Different values may be entered by the user. When the "M" value is omitted, the input fields will be empty even though default values have been defined.

The "T" value in `AD=MT` means that all lowercase input is translated to uppercase before further processing. This is important since the names in the demo database file have been defined completely in uppercase letters. When the "T" value is omitted, you have to enter all names completely in uppercase letters. Otherwise, the specified name will not be found.

"Start:" and "End:" are text fields (labels). They are specified in quotation marks.

#NAME-START and #NAME-END are data fields (input fields) in which the user can enter the desired starting name and ending name.

The slash (/) means that the subsequent fields are to be shown in a new line.

Your program should now look as follows:

```
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 Run the program.

The output shows the fields you have just defined.

```
Start: ADKINSON
End:   BENNETT
```

3 Use the default names and press ENTER.

The list of employees is now shown.

4 Press ENTER repeatedly until you return to the program editor, or enter EDIT at the MORE prompt.

5 Stow the program.

Designing a Map for User Input

You are now introduced to a different way of prompting the user for input. You will use the map editor to create a map which contains the same fields that you have previously defined in your program. A map is a separate object and is used to separate the user interface layout from the business logic of an application.

The map you will create now will look as follows:

```
XXXXXXXXXX                                                          TT:TT:TT


                          Start XXXXXXXXXXXXXXXXXXX

                            End XXXXXXXXXXXXXXXXXXX
```

The first line of the map contains system variables for the current date and time. There are two data fields (input fields) in which the user can specify a starting name and an ending name. The data fields are preceeded by text fields (labels).

The following steps are required for the above map:

- Creating a Map
- Defining Text Fields
- Defining Data Fields
- Specifying Names for Data Fields
- Adding System Variables
- Repositioning Fields
- Testing a Map

- Stowing a Map

Creating a Map

You will now invoke the map editor in which you will design your map.

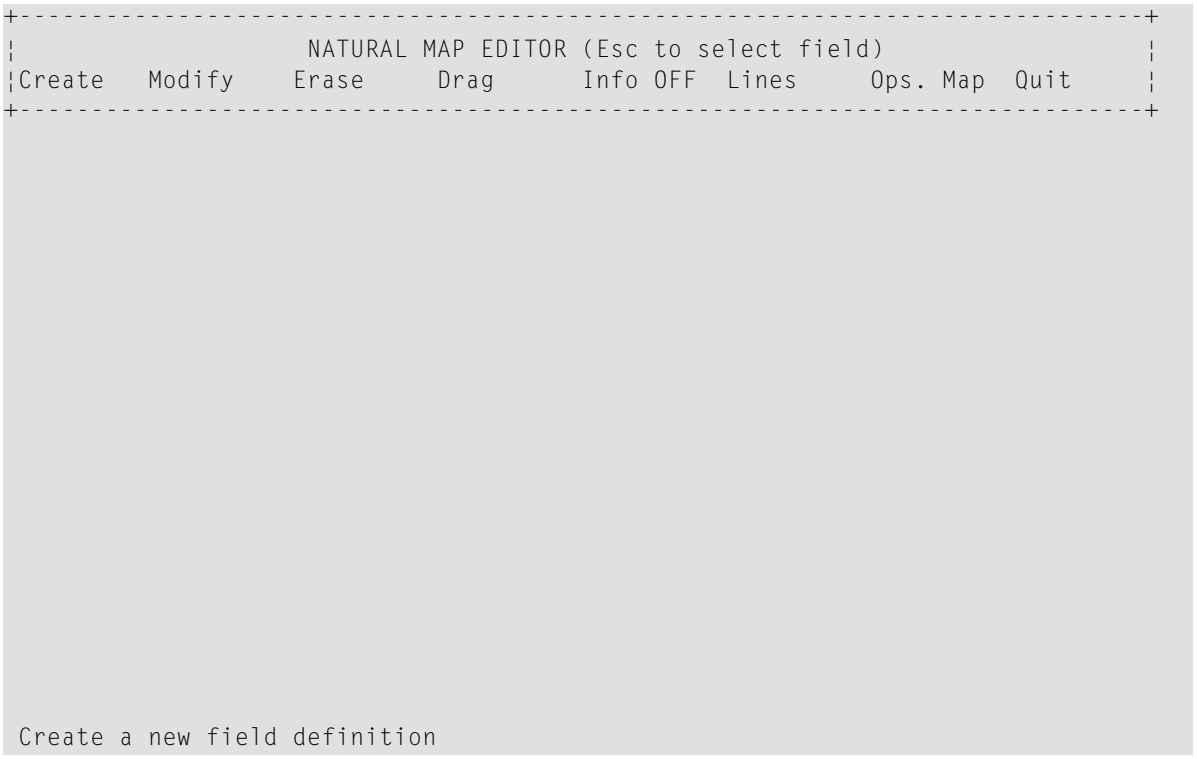
To create a map

- In the program editor's command line, enter one of the following:

```
EDIT MAP
```

```
E M
```

The **Natural Map Editor** menu appears.



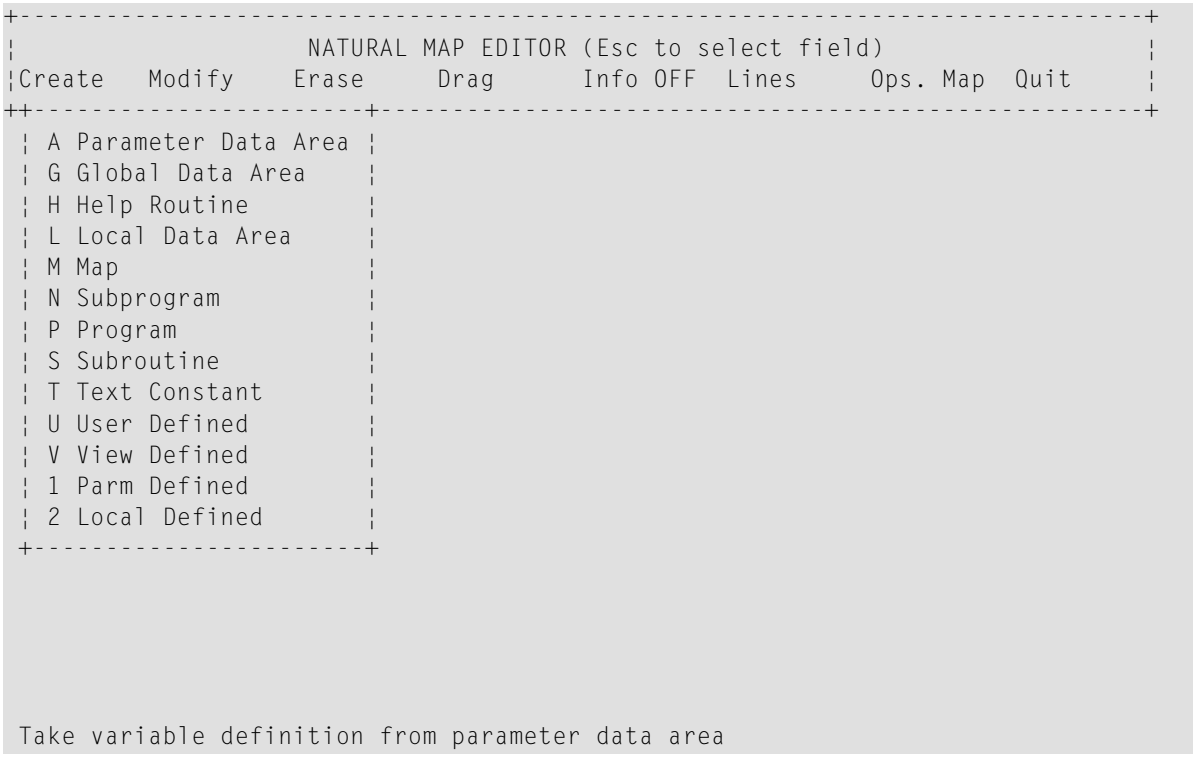
Defining Text Fields

You will now add two text fields (also called constants or labels) to the map.

To define the text fields

1 On the **Natural Map Editor** menu, select **Create** and press ENTER.

A window appears listing all items that you can create.



2 Use the arrow keys to select **Text Constant** and press ENTER.

Or:

Press T.

An empty screen appears. A message prompts you to position the cursor and to enter text.

3 Move the cursor to the first position of the fourth line (line and column number are shown at the bottom of the screen) and type in the following:

```
Start
```

Do not yet press ENTER.

4 Press PF2.

A window appears in which you can set the attributes for this text field.

```
Start


+-Text Attribute/Color Definition: --------------+
¦ Attribute: <No Attribute> Color: <No Color>    ¦
+-------------------------------------------------+
 MAP0083: Use cursor keys (UP, DOWN) for Attribute. (Esc=Cancel, Enter=OK)
```

The **Attribute** field is automatically selected.

Note: Using LEFT-ARROW and RIGHT-ARROW, you can toggle between the fields for attribute and color definition. This tutorial does not use colors.

5 Use UP-ARROW or DOWN-ARROW to scroll through the available attributes. When "Default" is shown in the window, press ENTER. "Default" means that the field is neither intensified nor in any way highlighted.

The field is now selected in the screen. This is indicated by a highlighted background.

6 Press ESC to redisplay the menu.

ESC is used as a toggle to display and hide the menu. The text field you have just defined is no longer shown. It has not been deleted. It has just disappeared behind the menu. When you press ESC once more, the menu disappears and your field is shown again.

7 Add a second text field in the same way as you have added the first text field (that is: define a second text constant and set the **Default** attribute). Place it in the line below the **Start** field (that is in line 5) and name it as follows:

```
End
```

8 When you have added the second text field, press ESC to redisplay the menu.

You should now see the new **End** field right below the menu.

Defining Data Fields

You will now add two data fields to the map. These are the input fields in which the user can specify the starting name and ending name.

To define the data fields

1 From the **Create** menu, choose **User Defined**.

The **Extended Field Editing** window appears:

2 Position the cursor behind the **Start** field (leave a blank space between text field and data field) and press ENTER.

The following window appears.

```
+--------------------+
¦ A Alphanumeric     ¦
¦ B Binary           ¦
¦ D Date             ¦
¦ F Floating Point   ¦
¦ I Integer          ¦
¦ L Logical          ¦
¦ N Numeric          ¦
¦ P Packed Numeric   ¦
¦ T Time             ¦
¦ * System           ¦
+--------------------+
```

3 Choose **A Alphanumeric**.

The selected format is shown in the **Format** field of the **Extended Field Editing** window. The cursor is automatically positioned in the **AL** field.

4 Enter "20" in the **AL** field.

This means that the data field is defined with a length of 20 characters.

5 Press TAB.

In the map, the length of the field is now indicated by "X" characters.

6 Press TAB repeatedly until the **AD** field is highlighted.

You will now define the I/O characteristics and the filler character. The following attribute definition characters are currently shown in the **AD** field:

```
ILMFHT'_'
```

The I/O characteristics are indicated by the letter "M". The filler character is indicated by `'_'`. This tutorial uses these default settings. The following steps are just provided to illustrate the usage of this function. At this point, you can also close the **Attribute Definition** window and the **Extended Field Editing** window as described further below.

Note: The sequence in which the attribute definition characters are arranged in the **AD** field may change, that is: an attribute may occur in a different position.

7 Press PF2 to edit the attribute definitions.

The **Attribute Definition** window appears.

```
+-Attribute Definition--+
¦Representation         ¦
¦Alignment              ¦
¦I/O Characteristics    ¦
¦Mandatory Characters   ¦
¦Length Characteristics ¦
¦Upper/Lower Case       ¦
¦Filler Character       ¦
+-----------------------+
```

8 Choose **I/O Characteristics**.

The following window appears.

```
+------------------------+
¦ A Input, non_protected ¦
¦ M Output, modifiable   ¦
¦ O Output, protected    ¦
+-Esc=Quit, Enter=Toggle-+
```

9 When the character "M" is already shown in the **AD** field, press the ESC key to leave this window.

Or:

When the character "M" is not shown in the **AD** field, choose **M Output, modifiable**.

Note: When you choose the same attribute several times, a toggle effect occurs: the attribute is then either defined or removed.

10 In the **Attribute Definition** window, choose **Filler Character**.

You are prompted to enter a new filler character.

The filler character is used to fill any empty positions in input fields in the map, allowing the user to see the exact position and length of a field when entering input.

11 Enter an underscore (_) at the current cursor position.

It is not necessary to press ENTER. The message disappears immediately after you have entered the underscore character.

12 Press ESC to close the **Attribute Definition** window.

13 Press ENTER to close the **Extended Field Editing** window and to store the field in the map.

Caution: When you press ESC instead of ENTER, your definitions for the new data field are not saved and the field is not stored in the map.

14 Add a second data field in the same way as described in the above steps: place it next to the **End** field, define a length of 20 characters and define the same attribute definition characters as described above.

Specifying Names for Data Fields

When you create a new data field for a user-defined variable, Natural assigns a field name to it. This field name contains a number. You have to adjust the names of the newly created fields to the variable names defined in your program.

You will now make sure that the same names are used as in your program: `#NAME-START` and `#NAME-END`. The output of these fields (that is: the user input) will be passed to the corresponding user-defined variables in your program.

To define names for the data fields

1 Use the arrow keys to select the data field for the starting name.

2 Press ESC to redisplay the menu.

3 In the menu, choose **Modify**.

The **Extended Field Editing** window appears.

```
+-Extended Field Editing-------------------------------------------------------+
¦Field : #1                                                                    ¦
¦Format: A Len: 20         AL: 20          PM:                  ZP: N  SG: N   ¦
¦Rules : O Rule Editing: N Array: None     Array Editing: N     Mode:  Undef   ¦
¦AD: ILMFHT'_'  CD:        CV:                                  DY: N  HE: N   ¦
¦EM:                                                                           ¦
+-------------------------------------------------------------------------------+
```

The **Field** field contains the field name that has been assigned by Natural: "#1".

4 In the **Field** field, enter "#NAME-START".

5 Press ENTER.

6 Repeat the above steps for the data field for the ending name and enter "#NAME-END" as the field name.

Adding System Variables

Natural system variables contain information about the current Natural session, such as the current library, user, or date and time. They may be referenced at any point within a Natural program. All system variables begin with an asterisk (*).

You will now add system variables for the date and time to the map. When the program is run, the current date and time will be displayed in the map.

To add system variables

1 Press ESC to redisplay the menu.

2 From the **Create** menu, choose **User Defined**.

The **Extended Field Editing** window appears.

3 Position the cursor in the top left corner of the screen (row 001, column 001) and press ENTER.

The following window appears.

```
+---------------------+
¦ A Alphanumeric      ¦
¦ B Binary            ¦
¦ D Date              ¦
¦ F Floating Point    ¦
¦ I Integer           ¦
¦ L Logical           ¦
¦ N Numeric           ¦
¦ P Packed Numeric    ¦
¦ T Time              ¦
¦ * System            ¦
+---------------------+
```

4 Choose *** System**.

The following window appears.

```
+----------------------------+
¦ *APPLIC-ID           A8    ¦
¦ *APPLIC-NAME         A32   ¦
¦ *COM                 A128  ¦
¦ *CONVID              I4    ¦
¦ *CPU-TIME            I4    ¦
¦ *CURRENT-UNIT        A32   ¦
¦ *CURSOR              N6    ¦
¦ *CURS-COL            P3    ¦
¦ *CURS-LINE           P3    ¦
¦ *DAT4D               A10   ¦
¦ *DAT4E               A10   ¦
¦ *DAT4I               A10   ¦
¦ *DAT4J               A7    ¦
¦ *DAT4U               A10   ¦
+----------------------------+
```

You can use DOWN-ARROW and UP-ARROW to scroll through the list of system variables.

5 Select ***DAT4I** and press ENTER.

A number of "X" characters is now shown.

6 Press ENTER to close the **Extended Field Editing** window.

7 Repeat the above steps and add the system variable ***TIMX**. Place it in the top right corner of the screen (row 001, column 070). **TT:TT:TT** will be shown for this system variable. Do not forget to press ENTER to close the **Extended Field Editing** window.

Repositioning Fields

You will now reposition the fields you have added.

To move a field

1 When the menu is currently shown, press ESC to hide it.

2 Use the arrow keys to select the data field for the starting name (which is indicated by "X" characters).

The data field is now highlighted.

3 Press ESC to redisplay the menu.

4 Choose **Drag**.

The menu disappears.

5 Use the arrow keys to move the highlighted field to the middle of the line.

6 Press ENTER to end drag mode.

7 Move the remaining text and data fields for the starting and ending names to the middle of the line as described above. Leave an empty line between the fields for the starting and ending names.

The map should now look as shown at the beginning of this section.

Testing a Map

You will now test your map to check whether it works as intended.

To test the map

1 Press ESC to redisplay the menu, and from the **Ops. Map** menu, choose **Test Map**.

The following output is shown.

```
2009-06-30                                                          13:39:55


                          Start  ____________________

                            End  ____________________
```

The input field for the starting name is automatically selected since it is the first input field in the map. Both input fields contain the filler character.

Note: When working in insert mode, the user has to delete the filler characters before it is possible to enter text. This is not necessary in overwrite mode which is the default.

2 Press ENTER to return to the map editor.

Stowing a Map

When the map has successfully been tested, you have to stow it so that it can be found by your program.

To stow the map

1 Press ESC to redisplay the menu.

2 From the **Ops. Map** menu, choose **Stow Map**.

The following window appears.

```
+--Stow Map As: ---+
¦Name...:          ¦
¦Library: TUTORIAL ¦
+------------------+
```

3 Enter "MAP01" as the name of the map and press ENTER.

Invoking the Map from Your Program

Once a map has been stowed, it can be invoked by a Natural program using a `WRITE` or `INPUT` statement.

To invoke the map from your program

1 Press ESC to redisplay the menu and choose **Quit**.

Natural's main menu is shown again with the **Direct Command** window prompting for input.

2 Return to the program editor by entering one of the following.

```
EDIT PGM01
```

```
E PGM01
```

3 Replace the previously defined `INPUT` lines with the following line:

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

This will invoke the map you have just designed.

The map name must be enclosed in single quotation marks to distinguish the map from a user-defined variable.

Your program should now look as follows:

```
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
```

4 Run the program.

Your map is now shown.

5 Press ENTER repeatedly until you return to the program editor, or enter `EDIT` at the `MORE` prompt.

6 Stow the program.

Ensuring that an Ending Name is Always Used

As your program is coded now, no data will not be found if an ending name is not specified.

You will now remove the initial values for the starting name and ending name; then the user always has to specify these names. To ensure that an ending name is always used, even if it has not been specified by the user, you will add a corresponding statement.

To use the ending name

1 In the `DEFINE DATA` block, remove the default values (`INIT`) for the fields `#NAME-START` and `#NAME-END` so that the corresponding lines look as follows:

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

2 Insert the following below `INPUT USING MAP 'MAP01'`:

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

When the `#NAME-END` field is blank (that is: when an ending name has not been entered by the user), the starting name is automatically used as the ending name.

Note: Instead of using the statement `MOVE #NAME-START TO #NAME-END` it is also possible to use the following variant of the `ASSIGN` or `COMPUTE` statement: `#NAME-END := #NAME-START`.

Your program should now look as follows:

```
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 Run the program.

4 In the resulting map, enter "JONES" in the field which prompts for a starting name and press ENTER.

In the resulting list, only the employees with the name "Jones" are now shown.

5 Press ENTER to return to the program editor.

6 Stow the program.

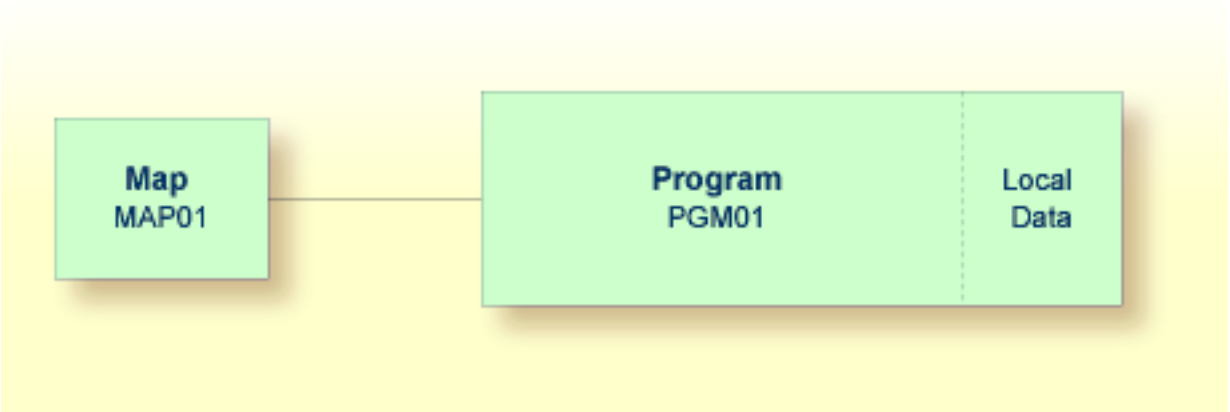
You can now proceed with the next exercises: ***Loops and Labels***.

6 Loops and Labels

- Allowing Repeated Usage 52
- Displaying a Message Indicating that Information was not Found 54

You will now enhance your program by adding loops and labels.

When you have completed the exercises below, your sample application will still consist of the same modules as in the previous chapter:



Allowing Repeated Usage

As it is now, the program terminates after it has displayed the map and has shown the list. So that the user can display a new employees list immediately, without restarting the program, you will now put the corresponding program code into a `REPEAT` loop.

To define a repeat loop

1 Insert the following below `END-DEFINE`:

```
RP1. REPEAT
```

`REPEAT` defines the start of the repeat loop. `RP1.` is a label which is used when leaving the repeat loop (this is defined below).

2 Define the end of the repeat loop by inserting the following before the `END` statement:

```
END-REPEAT
```

3 Insert the following below `INPUT USING MAP 'MAP01'`:

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

The IF statement, which must be ended with END-IF, checks the content of the #NAME-START field. When a dot (.) is entered in this field, the ESCAPE BOTTOM statement is used to leave the loop. Processing will continue with the first statement following the loop (which is END in this case).

By assigning a label to the loop (here RP1.), you can refer to this specific loop in the ESCAPE BOTTOM statement. Since loops may be nested, you should specify which loop you want to leave. Otherwise, the program will only leave the innermost active loop.

Your program should now look as follows:

```
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: For better readability, the content of the REPEAT loop has been indented.

4 Run the program.

5 In the resulting map, enter "JONES" in the field which prompts for a starting name and press ENTER.

In the resulting list, the employees with the name "Jones" are shown. Press ENTER. Due to the REPEAT loop, the map is shown again. Now you can also see that "JONES" has been entered as the ending name.

6 To leave the map, enter a dot (.) in the field which prompts for a starting name and press ENTER. Do not forget to delete the remaining characters of the name which is still shown in this field.

7 Stow the program.

Displaying a Message Indicating that Information was not Found

You will now define the message that is to be displayed when the user enters a starting name which cannot be found in the database.

To define the message that is to be displayed when the specified employee cannot be found

1 Add the label RD1. to the line containing the READ statement so that it looks as follows:

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

2 Insert the following below END-READ:

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

To check the number of records found in the READ loop, the system variable *COUNTER is used. If its contents equals 0 (that is: an employee with the specified name has not been found), the message defined with the REINPUT statement is displayed at the bottom of your map.

To identify the READ loop, you assign a label to it (here RD1.). Since a complex database access program can contain many loops, you have to specify the loop to which you refer.

Your program should now look as follows:

```
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 Run the program.

4 In the resulting map, enter a starting name which is not defined in the demo database (for example, "XYZ") and press ENTER.

Your message should now appear in the map.

5 To leave the map, enter a dot (.) in the field which prompts for a starting name and press ENTER. Do not forget to delete the remaining characters of the name which is still shown in this field.

6 Stow the program.

You can now proceed with the next exercises: ***Inline Subroutines***.

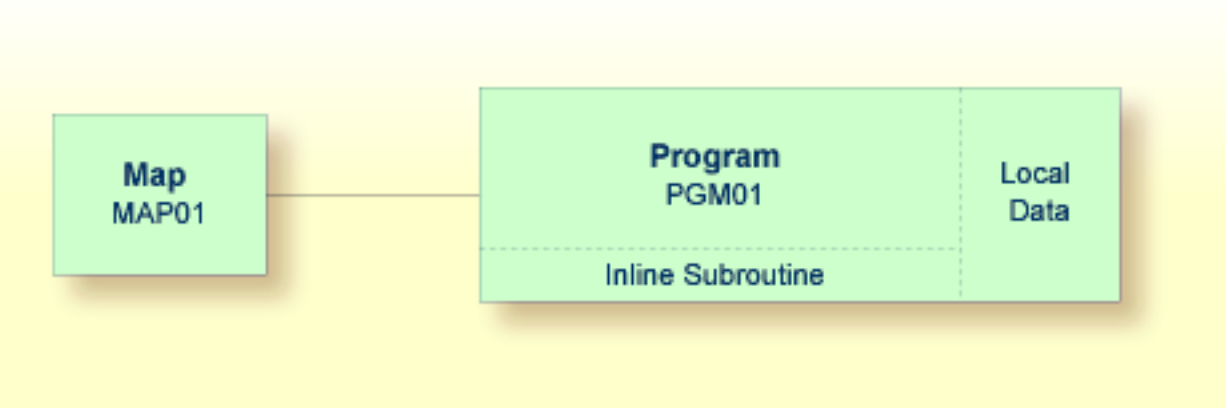
7 Inline Subroutines

- Defining the Inline Subroutine 58
- Performing the Inline Subroutine 59

Natural distinguishes two types of subroutines: inline subroutines which are defined directly in the program and external subroutines which are stored as separate objects outside the program (this is explained later in this tutorial).

You will now add an inline subroutine to your program which moves an asterisk (*) to the new user-defined variable named #MARK. This subroutine will be invoked when an employee has 20 days of leave or more.

When you have completed the exercises below, your sample application will be structured as follows:



Defining the Inline Subroutine

You will now add the subroutine to your program.

To define the subroutine

1 Insert the following below the user-defined variable #NAME-END:

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

This variable will be used by the subroutine. Therefore, it has to be defined first.

2 To define the subroutine, insert the following before the END statement:

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

When performed, this subroutine moves an asterisk (*) to #MARK.

Note: Instead of using the statement MOVE '*' TO #MARK it is also possible to use the following variant of the ASSIGN or COMPUTE statement: #MARK := '*'.

3 Modify the DISPLAY statement as follows:

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

This displays a new column in your output. Its heading is ">=20". The column will contain an asterisk (*) if the corresponding employee has 20 days of leave or more.

Performing the Inline Subroutine

Now that you have defined the inline subroutine, you can specify the corresponding code for performing it.

To perform the inline subroutine

1 Insert the following before the DISPLAY statement:

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

When an employee is found who has 20 days of leave or more, the new subroutine named MARK-SPECIAL-EMPLOYEES is performed. When an employee has less than 20 days of leave, the content of #MARK is reset to blank.

Your program should now look as follows:

```
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 Run the program.

3 In the resulting map, enter "JONES" and press ENTER.

The list of employees should now contain the additional column.

4 To return to the program editor, enter `EDIT` at the `MORE` prompt.

5 Stow the program.

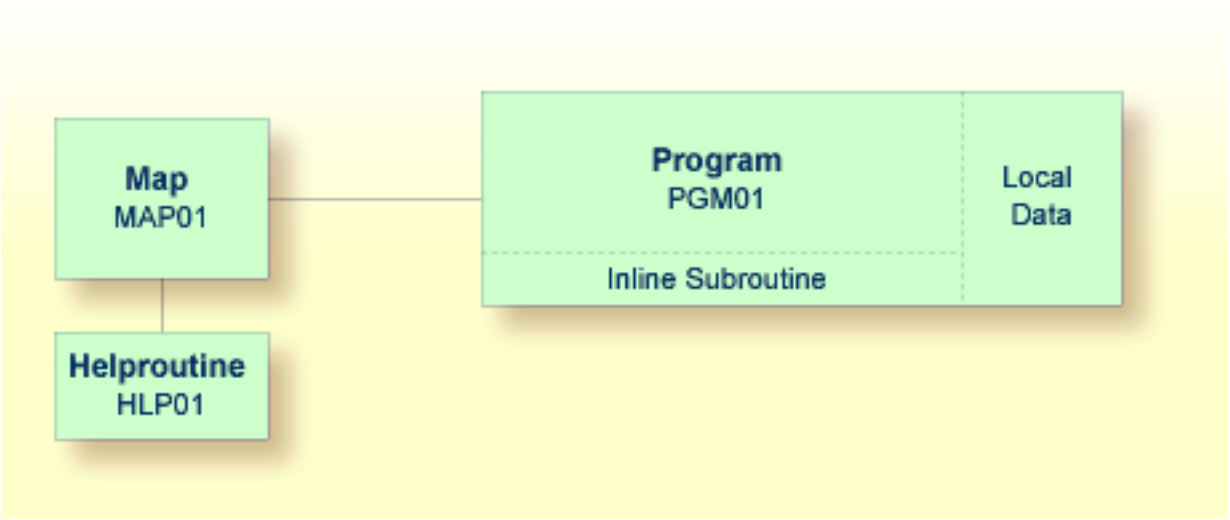
You can now proceed with the next exercises: ***Processing Rules and Helproutines***.

8 Processing Rules and Helproutines

- Defining a Processing Rule 64
- Defining a Helproutine 67

Processing rules and helproutines are defined for fields in a map.

When you have completed the exercises below, your sample application will consist of the following modules (a processing rule cannot be defined as a separate module; it is always part of a map):



Defining a Processing Rule

You will now define the message that is to be displayed when the user presses ENTER without specifying a starting name.

To define a processing rule

1 Return to the map editor by entering the following.

```
EDIT MAP01
```

2 Use the arrow keys to select the input field for the starting name. Press ESC to redisplay the menu and choose **Modify**.

The **Extended Field Editing** window appears.

```
+-Extended Field Editing-------------------------------------------------------+
¦Field : #NAME-START                                                          ¦
¦Format: A Len: 20         AL: 20          PM:                  ZP: N  SG: N  ¦
¦Rules : 0 Rule Editing: N Array: None     Array Editing: N     Mode:  User   ¦
¦AD: ILMFHT'_'  CD:        CV:                                  DY: N  HE: N  ¦
¦EM:                                                                          ¦
+------------------------------------------------------------------------------+
```

3 Use TAB to move to the field **Rule Editing**. Either enter "Y" in this field or press PF2.

The following screen appears.

```
+-Current Field: #NAME-START---------------------------------------------------+
¦                      R U L E   E D I T I N G (Esc = Quit)                   ¦
¦Rules                                  Fields                                ¦
+------------------------------------------------------------------------------+

                           End XXXXXXXXXXXXXXXXXXX









 Create or modify a rule for this field
```

4 Choose **Rules** and then **<CREATE>**.

An empty editor window appears.

5 Enter the following processing rule (use the line command `I` to insert blank lines):

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

The ampersand (&) in the processing rule will dynamically be replaced with the name of the field. In this case, it will be replaced with `#NAME-START`. If `#NAME-START` is blank, the message defined with the `REINPUT` statement is displayed.

MARK is an option of the REINPUT statement. Its syntax is MARK *fieldname*. MARK specifies the field in which the cursor is to be placed when the REINPUT statement is executed. In this case, the cursor will be placed in the #NAME-START field.

6 Enter the following in the command line:

```
P=1
```

The rank defines the sequence in which the rules for the different fields are to be processed. All rules with rank 1 are processed first, followed by those with rank 2, etc.

```
EDIT:                                    S 02- ---------------Columns 001 074
Rank:  NEW RULE      Rule:                                       Typ: R  Mode: S
Cmnd=> p=1                                                    Scroll==> CSR
****** ******************************** top of data ******************************
000001 IF & = ' ' THEN
000002   REINPUT 'PLEASE ENTER A STARTING NAME.'
000003   MARK *&
000004 END-IF
****** ***************************** bottom of data ****************************









PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12--
P     U     End   P*    Rfind Rchng Up    Down        Right Left  Home
```

7 Press ENTER to save your input.

The following screen appears again. Your new rule with the rank 1 is now shown below **<CREATE>**. When you select this rule, the first lines of your code are shown at the bottom of the screen.

```
+-Current Field: #NAME-START-------------------------------------------------+
¦                      R U L E   E D I T I N G (Esc = Quit)                  ¦
¦Rules                                  Fields                               ¦
+--------+-----------------------------------------------------------------+
¦<CREATE>¦
¦       1¦             End XXXXXXXXXXXXXXXXXXX
+--------+
```

```
0010 IF & = ' ' THEN
0020   REINPUT 'PLEASE ENTER A STARTING NAME.'
0030   MARK *&

 Create or modify a rule for this field
```

8 Press ESC repeatedly until the Natural Map Editor menu is shown again.

9 Test the map.

10 In the resulting output, enter any starting name and press ENTER.

The output screen is closed.

11 Test the map once more. Do not enter a name and press ENTER.

The message defined with the processing rule should now appear in the map.

12 To leave the output screen, enter a dot (.) in the field which prompts for a starting name and press ENTER.

13 Stow the map.

Defining a Helproutine

A helproutine is displayed when the user presses the help key when the cursor is on the input field for the starting name.

You will first define the helproutine and then associate it with a specific field.

To create a helproutine

1 From the Natural Map Editor menu, choose **Quit**.

Natural's main menu is shown again with the **Direct Command** window prompting for input.

2 Enter one of the following:

```
EDIT PROGRAM
```

```
E P
```

An empty editor appears.

3 Enter the following:

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

4 Change the program to a helproutine by entering the following in the program editor's command line:

```
SET TYPE H
```

where "H" denotes helproutine.

5 Stow the helproutine and name it `HLP01`.

```
STOW HLP01
```

To associate the helproutine with a field on the map

1 Return to the map editor by entering the following in the command line of the screen in which you have just entered the helproutine.

```
E MAP01
```

2 Select the data field for the starting name, press ESC to display the Natural Map Editor menu and choose **Modify**.

The **Extended Field Editing** window is displayed for the field.

3 Use TAB to move to the field **HE**. Either enter "Y" in this field or press PF2.

A window appears prompting for the name of the helproutine.

4 In the **HE** field enter "'HLP01'" (including the single quotation marks).

This is the name under which you have saved your helproutine.

```
XXXXXXXXXX                                                          TT:TT:TT


                         Start XXXXXXXXXXXXXXXXXXX

                           End XXXXXXXXXXXXXXXXXXX




     +-HE-------------------------------------------------------------------+
     ¦HE: 'HLP01'                                                          ¦
     ¦                                                                     ¦
     +---------------------------------------------------------------------+

+-Extended Field Editing-------------------------------------------------------+
¦Field : #NAME-START                                                          ¦
¦Format: A Len: 20         AL: 20          PM:                  ZP: N  SG: N  ¦
¦Rules : 1 Rule Editing: Y Array: None     Array Editing: N     Mode:  User   ¦
¦AD: MILFHT'_'  CD:        CV:                                  DY: N  HE: Y  ¦
¦EM:                                                                          ¦
+-----------------------------------------------------------------------------+
 Helproutine and Parameters
```

5 Press ENTER twice to save your changes and close all windows.

6 Test the map.

7 In the resulting output, enter a question mark (?) in the input field for the starting name.

The help text you have defined is shown.

8 Press ENTER to return to the map.

9 To leave the map, enter a dot (.) in the field which prompts for a starting name and press ENTER.

10 Stow the map.

11 From the Natural Map Editor menu, choose **Quit**.

Natural's main menu is shown again with the **Direct Command** window prompting for input.

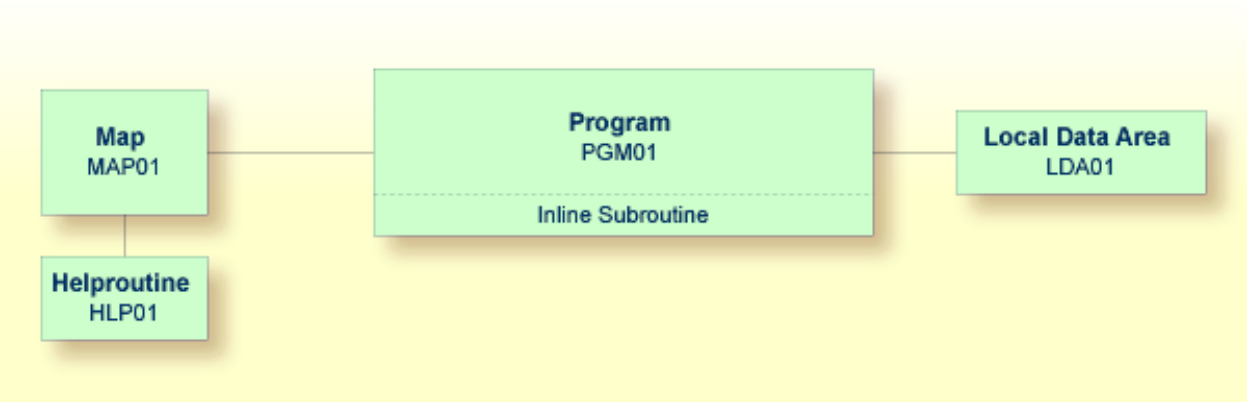
You can now proceed with the next exercises: ***Local Data Areas***.

9 Local Data Areas

- Creating a Local Data Area 72
- Defining Data Fields 73
- Importing the Required Data Fields from a DDM 75
- Referencing the Local Data Area from Your Program 78

Currently, the fields used by your program are defined within the `DEFINE DATA` statement in the program itself. It is also possible, however, to place the field definitions in a local data area (LDA) outside the program, with the program's `DEFINE DATA` statement referencing this local data area by name. For reusability and for a clear application structure, it is usually better to define fields in data areas outside the programs.

You will now relocate the information from the `DEFINE DATA` statement to a local data area. When you have completed the exercises below, your sample application will consist of the following modules:



Creating a Local Data Area

You will now invoke the data area editor in which you will specify the required fields.

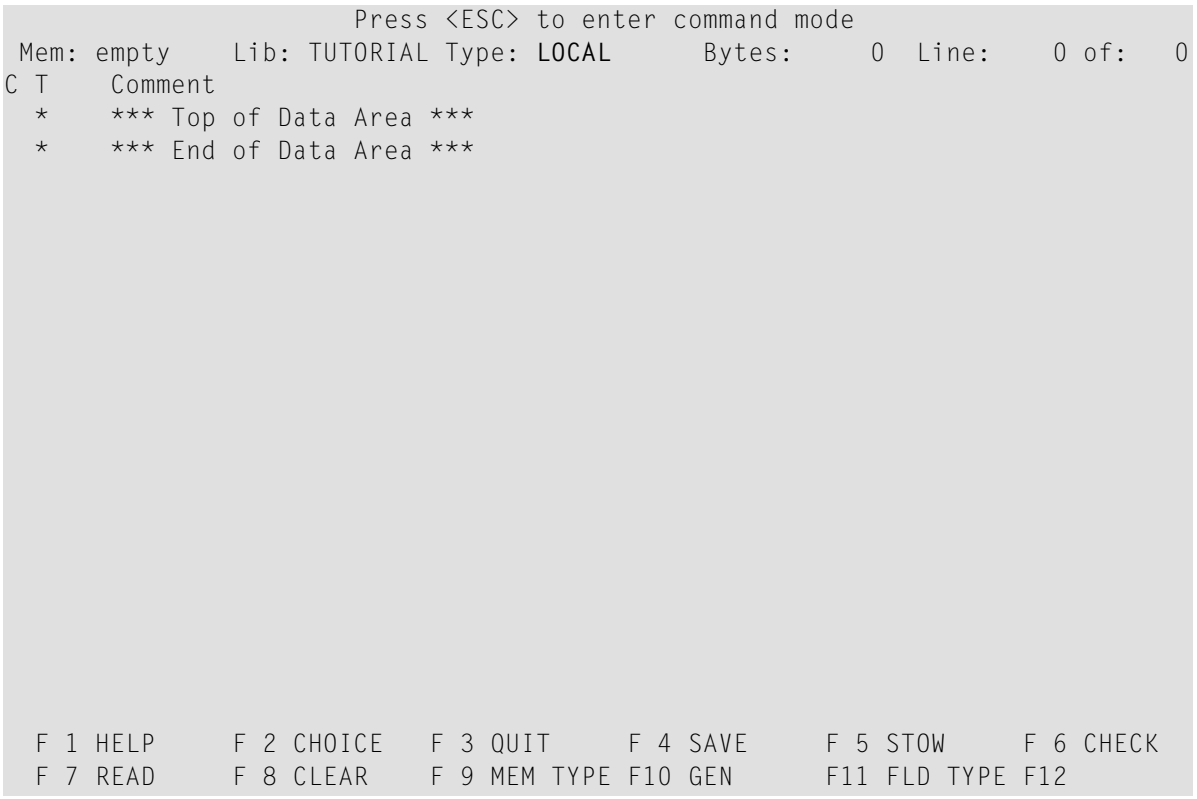
To invoke the data area editor

- In the **Direct Command** window, enter one of the following:

  ```
  EDIT LOCAL
  ```

  ```
  E L
  ```

  The data area editor appears. The object type has been set to "Local". This is indicated at the top of the screen.



By default, the data area editor is in edit mode when you invoke it. To toggle from edit mode to command mode (and vice versa), you press ESC. When you leave the editor, the current mode will be resumed when you open the editor the next time.

Defining Data Fields

You will now define the following fields:

Level	Name	Format	Length
1	#NAME-START	A	20
1	#NAME-END	A	20
1	#MARK	A	1

These are the user-defined variables which you have previously defined in the `DEFINE DATA` statement.

To define the data fields

1 Make sure that the first entry, which indicates the top of the data area, is selected in the editor.

Tip: If it is not possible to move the cursor and thus to highlight a line, you are in command mode. In this mode, the command line is shown at the top of the screen. Press ESC to switch to edit mode.

2 Enter the following line command in the first column of the selected line:

```
I
```

You need not press ENTER.

The following window appears.

```
+-----------------------+
¦ D Data Field          ¦
¦ B Block               ¦
¦ C Constant            ¦
¦ H Handle              ¦
¦ S Structure           ¦
¦ * Comment             ¦
+-----------------------+
```

3 Choose **Data Field**.

The **Data Field Definition** window appears.

```
+-------------------- Data Field Definition ----------------------+
¦ Level:           1                                              ¦
¦ Field Name:                                                     ¦
¦ Field Format:                                                   ¦
¦ Field Length:                                                   ¦
¦ Arraydefinition:                                                ¦
¦ Edit Mask:                                                      ¦
¦                                                                 ¦
¦ Header Definition:                                              ¦
¦                                                                 ¦
¦ Initialization:                                                 ¦
¦ Value Clause:                                                   ¦
¦ Optional Param:  N                                              ¦
¦ Comment:                                                        ¦
+-------------------------------------------------------------------+
```

4 Specify all required information for the first field (#NAME-START) as listed in the above table. Use the arrow keys to move from one field to the next.

5 When all information for this field has been specified, press ENTER to save this information.

The **Data Field Definition** window remains open. The input fields are empty again and you can define a new data field.

6 Specify all required information for the remaining fields (#NAME-END and #MARK) as described above.

7 When all fields have been added, press ESC.

The fields you have defined are now shown in the editor.

```
                      Press <ESC> to enter command mode
 Mem:          Lib: TUTORIAL Type: LOCAL      Bytes:   291  Line:    3 of:   3
C T L  Name of Datafield                F     Length Index/Comment             M
  *    *** Top of Data Area ***
     1 #NAME-START                      A         20
     1 #NAME-END                        A         20
     1 #MARK                            A          1
  *    *** End of Data Area ***
  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

Tip: If you notice that you have made a mistake in a field definition, you can use the line command E to edit the selected field or the line command D to delete the selected field.

Importing the Required Data Fields from a DDM

You will now import the same data fields which you have previously defined in the program's DEFINE DATA statement. The fields are read directly from a Natural data view into the data area editor. A data view references database fields defined in a data definition module (DDM).

In the data area editor, the imported data fields will be inserted below the currently selected data field.

To import data fields from a DDM

1 Enter the following line command in the first column of the #MARK line.

```
V
```

The **View Definition** window appears.

```
                       Press <ESC> to enter command mode
 Mem:          Lib: TUTORIAL Type: LOCAL      Bytes:   291  Line:    3 of:   3
C T L  Name of Datafield                F     Length Index/Comment             M
  *    *** Top of Data Area ***
     1 #NAME-START                      A         20
     1 #NAME-END                        A         20
V    1 #MARK                            A          1
  *    *** End of Data Area ***

           +------------------- View Definition --------------------+
           ¦Name of View:                                           ¦
           ¦Name of DDM:                                            ¦
           ¦Comment:                                                ¦
           +----------------------------------------------------------+









  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

2 Enter the following information and press ENTER:

```
+------------------- View Definition --------------------+
¦Name of View:          EMPLOYEES-VIEW                   ¦
¦Name of DDM:           EMPLOYEES                        ¦
¦Comment:                                                ¦
+----------------------------------------------------------+
```

A window appears showing the fields of the specified DDM.

```
                      Press <ESC> to enter command mode
 Mem:          Lib: TUTORIAL Type: LOCAL      Bytes:   291  Line:    3 of:   3
C T L  Name of Datafield                F     Length Index/Comment             M
  *    *** Top of Data Area ***
     1 #NAME-START                      A         20
     1 #NAME-END                        A         20
V    1 #MARK                            A          1
  *    *** End of Data Area ***




+------------------------------- DDM: EMPLOYEES --------------------------------+
¦    1  AA  PERSONNEL-ID                A          8    D                      ¦
¦  G 1  AB  FULL-NAME                                                          ¦
¦    2  AC  FIRST-NAME                  A         20  N                       ¦
¦    2  AD  MIDDLE-I                    A          1  N                       ¦
¦    2  AE  NAME                        A         20    D                     ¦
¦    1  AD  MIDDLE-NAME                 A         20  N                       ¦
+------------------------------------------------------------------------------+
     HD=PERSONNEL/ID


  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

3 Mark the following fields by entering an "X" in the first column:

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

Use the arrow keys to scroll through the DDM.

Note: The field `PERSONNEL-ID` will be used later when you create the subprogram.

4 After you have marked all required fields, press ENTER.

The local data area should now look as follows (use the arrow keys to scroll to the top of the data area):

```
                      Press <ESC> to enter command mode
 Mem:          Lib: TUTORIAL Type: LOCAL      Bytes:   970  Line:    0 of:  10
C T    Comment
  *    *** Top of Data Area ***
     1 #NAME-START                       A          20
     1 #NAME-END                         A          20
     1 #MARK                             A           1
  V  1 EMPLOYEES-VIEW                                  EMPLOYEES
     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
  *    *** End of Data Area ***









  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

The **T** column indicates the type of the variable. The view is indicated by a "V" and each group is indicated by a "G".

5 Press ESC to enter command mode.

6 Stow the local data area and name it "LDA01".

```
STOW LDA01
```

Referencing the Local Data Area from Your Program

Once a local data area has been stowed, it can be referenced by a Natural program.

You will now change the `DEFINE DATA` statement your program so that it uses the local data area that you have just defined.

To use the local data area in your program

1 Return to the program editor by entering the following in the command line of the data area editor.

```
E PGM01
```

2 In the DEFINE DATA statement, delete all variables between LOCAL and END-DEFINE (use the line command D).

3 Add a reference to your local data area by modifying the LOCAL line as follows:

```
LOCAL USING LDA01
```

Your program should now look as follows:

```
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 Run the program.

5 To confirm that the results are the same as before (when the `DEFINE DATA` statement did not reference a local data area), enter "JONES" as the starting name and press ENTER.

6 To return to the program editor, enter `EDIT` at the `MORE` prompt.

7 Stow the program.

You can now proceed with the next exercises: ***Global Data Areas***.

10 Global Data Areas

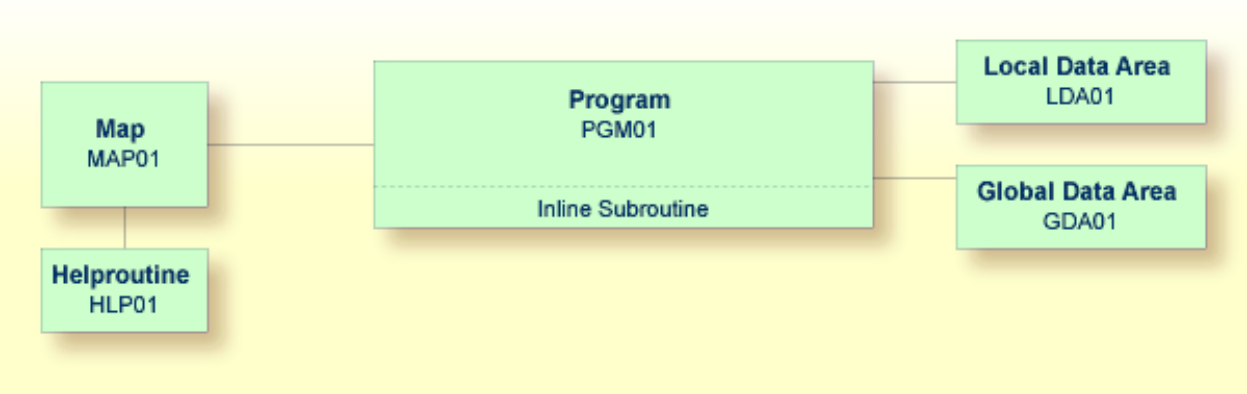
- Creating a Global Data Area from an Existing Local Data Area 82
- Adapting the Local Data Area 84
- Referencing the Global Data Area from Your Program 85

Data defined in a global data area (GDA) can be shared by multiple programs, external subroutines and helproutines.

Any modification of a data element value in a global data area affects all Natural objects that reference this global data area. Therefore, if you change the source of a global data area, you have to stow all previously created Natural objects that reference this global data area once more. The sequence in which objects are stowed is important. You must first stow the global data area and then the program. If you stow the program first and then the global data area, the program cannot be stowed because new elements in the global data area cannot be found.

You will now create a global data area which will be shared by your program and an external subroutine that you will create later. As the basis for your global data area, you will use some of the information from the local data area you have just created.

When you have completed the exercises below, your sample application will consist of the following modules:



Creating a Global Data Area from an Existing Local Data Area

You can create a new data area from an existing data area by editing it and saving it under a different name and with a different type. The original data area remains unchanged, and the new data area can be edited. Since the fields #NAME-START and #NAME-END are not required in the global data area, you will remove them.

To create the global data area

1 Return to your local data area by entering the following in the command line of the program editor.

```
E LDA01
```

2 To save the data area under a new name, enter the following in the command line of the data area editor.

```
SA GDA01
```

The current data area is saved with the new name `GDA01`. The local data area named `LDA01` is still shown in the data area editor.

3 Load `GDA01` into the data area editor by entering the following command:

```
E GDA01
```

4 To change the local data area into a global data area, enter the following command:

```
SET TYPE G
```

where "G" denotes global data area.

The object type changes to "Global". This is indicated at the top of the screen.

5 Press ESC to enter edit mode. Use the line command `D` to delete the following fields:

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

6 The global data area should now look as follows:

```
                       Press <ESC> to enter command mode
 Mem: GDA01    Lib: TUTORIAL Type: GLOBAL     Bytes:   351  Line:    1 of:   8
C T L  Name of Datafield                F     Length Index/Comment             M
  *    *** Top of Data Area ***
     1 #MARK                            A          1
  V  1 EMPLOYEES-VIEW                                EMPLOYEES
     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
  *    *** End of Data Area ***









  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

7 Stow the global data area.

Adapting the Local Data Area

The fields contained in the global data area are no longer required in the local data area. Therefore, you will now remove all fields except #NAME-START and #NAME-END from the local data area.

To remove the fields

1 Return to your local data area by entering the following in the command line of the data area editor:

```
E LDA01
```

2 Use the line command D to delete all fields except #NAME-START and #NAME-END.

When you delete the top-level entry for the view (indicated by a "V" in front of the view name), all fields belonging to this view are automatically deleted.

3 Stow the modified local data area.

The local data area should now look as follows:

```
Command:
 Mem: LDA01    Lib: TUTORIAL Type: LOCAL      Bytes:    85  Line:      of:   2
C T    Comment
  *    *** Top of Data Area ***
     1 #NAME-START                      A         20
     1 #NAME-END                        A         20
  *    *** End of Data Area ***
```

```
  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

Referencing the Global Data Area from Your Program

Once a global data area has been stowed, it can be referenced by a Natural program.

You will now change the `DEFINE DATA` statement in your program so that it also uses the global data area that you have just defined.

To use the global data area in your program

1 Return to the program editor by entering the following in the command line of the data area editor.

```
E PGM01
```

2 Insert the following in the line above `LOCAL USING LDA01`:

```
GLOBAL USING GDA01
```

A global data area must always be defined before a local data area. Otherwise, an error occurs.

Your program should now look as follows:

```
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 Run the program.

4 To confirm that the results are the same as before (when the `DEFINE DATA` statement did not reference a global data area), enter "JONES" as the starting name and press ENTER.

5 To return to the program editor, enter `EDIT` at the `MORE` prompt.

6 Stow the program.

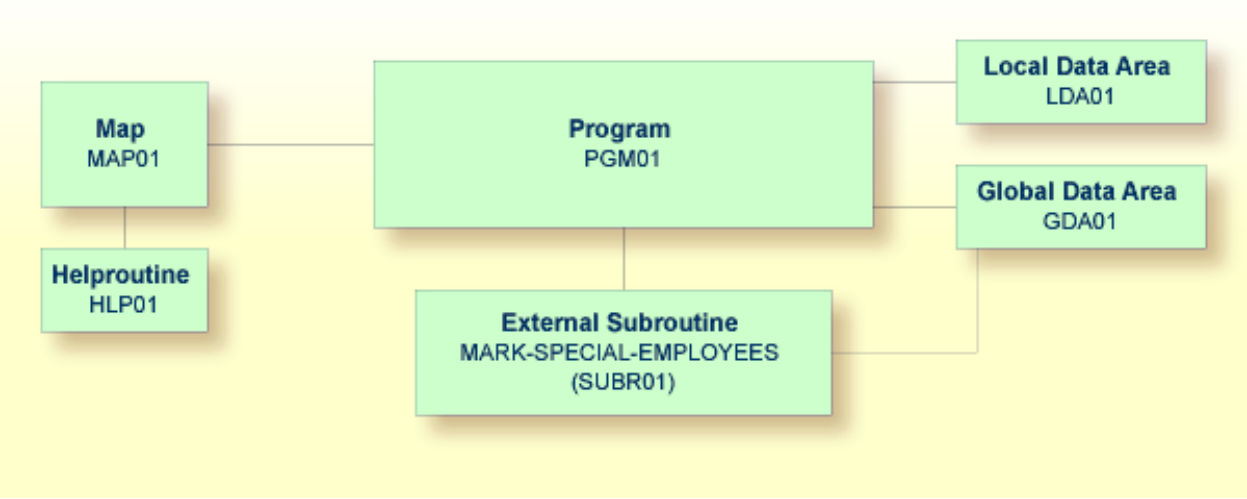
You can now proceed with the next exercises: ***External Subroutines***.

11 External Subroutines

- Creating an External Subroutine 88
- Referencing the External Subroutine from Your Program 89

Until now, the subroutine `MARK-SPECIAL-EMPLOYEES` has been defined within the program using a `DEFINE SUBROUTINE` statement. You will now define the subroutine as a separate object external to the program.

When you have completed the exercises below, your sample application will consist of the following modules:



Creating an External Subroutine

Since the existing code from the program will be reused in the external subroutine, you will now save the program under a new name, change its type to subroutine and delete all lines that are not required.

The `DEFINE SUBROUTINE` statement of the external subroutine is coded in the same way as the inline subroutine in the program.

To create an external subroutine

1 Enter the following in the command line of the program editor.

```
SA SUBR01
```

The current program is saved with the new name `SUBR01`. The program is still shown in the editor.

2 Load the newly created object into the editor by entering the following command:

```
E SUBR01
```

The object type is still program.

3 To change the program into an external subroutine, enter the following command:

```
SET TYPE S
```

where "S" denotes subroutine.

The object type which is shown in the screen changes to "Subroutine".

4 Use the line command `D` to delete all lines except the following:

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

It is also possible to delete a block of text. To do so, you have to proceed as follows:

1. At the beginning of the first line of the block, enter the line command `.X`.
2. At the beginning of the last line of the block, enter the line command `.Y`.
3. Press ENTER.

   The block of lines to be deleted is now marked with ".X" and ".Y". (If you want to remove the marks, you can enter `RESET` in the command line.)
4. To delete the marked block, enter `DX-Y` in the command line.

5 Stow the subroutine.

Referencing the External Subroutine from Your Program

The `PERFORM` statement invokes both internal and external subroutines. When an internal subroutine is not found in the program, Natural automatically tries to perform an external subroutine with the same name. Note that Natural looks for the name that has been defined in the subroutine code (that is: the subroutine name), not for the name that you have specified when saving the subroutine (that is: the Natural object name).

Now that you have defined an external subroutine, you have to remove the inline subroutine (which has the same name as the external subroutine) from your program.

To use the external subroutine in your program

1 Return to the program editor by entering the following in the command line of the editor in which the subroutine is currently shown.

```
E PGM01
```

2 Remove the following lines:

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

Your program should now look as follows:

```
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 Run the program.

4 Enter "JONES" as the starting name and press ENTER.

The resulting list should still show an asterisk for each employee who has 20 days of leave and more.

5 To return to the program editor, enter `EDIT` at the `MORE` prompt.

6 Stow the program.

You can now proceed with the next exercises: ***Subprograms***.

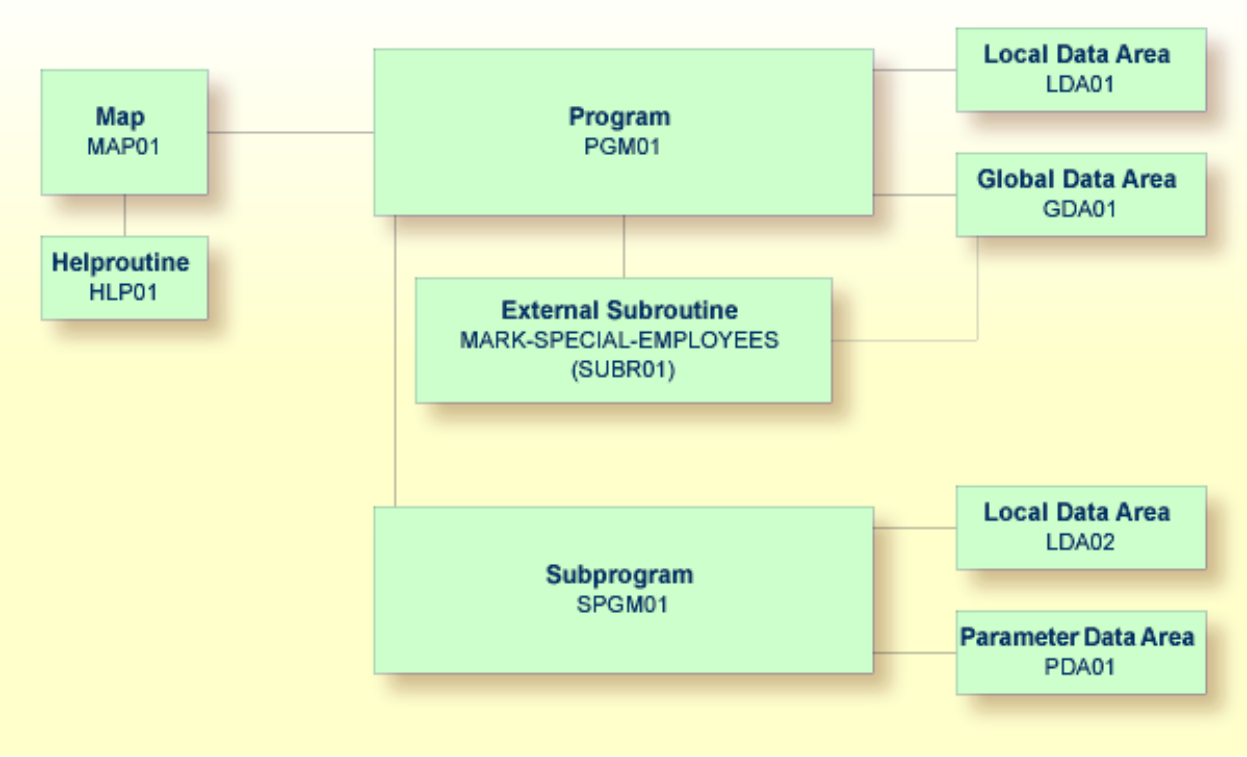
12 Subprograms

- Modifying the Local Data Area 94
- Creating a Parameter Data Area from an Existing Local Data Area 96
- Creating Another Local Data Area Containing a Different View 97
- Creating a Subprogram 99
- Referencing the Subprogram from Your Program 100

You will now expand your program to include a `CALLNAT` statement that invokes a subprogram. In the subprogram, the employees identified from the main program will be the basis of a `FIND` request to the `VEHICLES` file which is also part of the demo database. As a result, your output will contain vehicles information from the subprogram as well as employees information from the main program.

The new subprogram requires the creation of an additional local data area and a parameter data area.

When you have completed the exercises below, your sample application will consist of the following modules:



Modifying the Local Data Area

You will now add more fields to the local data area that you have previously created. These fields will be used by the subprogram that you will create later.

To add more fields to the local data area

1 Return to your local data area.

```
E LDA01
```

2 If required, press ESC to switch to edit mode.

3 Define the following fields below #NAME-END (use the line command I and from the resulting window, choose **Data Field**):

Level	Name	Format	Length
1	#PERS-ID	A	8
1	#MAKE	A	20
1	#MODEL	A	20

When all fields have been added, press ESC.

The local data area should now look as follows:

```
                       Press <ESC> to enter command mode
 Mem: LDA01    Lib: TUTORIAL Type: LOCAL      Bytes:   376  Line:    5 of:   5
C T L  Name of Datafield                F     Length Index/Comment             M
  *    *** Top of Data Area ***
     1 #NAME-START                      A         20
     1 #NAME-END                        A         20
     1 #PERS-ID                         A          8
     1 #MAKE                            A         20
     1 #MODEL                           A         20
  *    *** End of Data Area ***
  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

4 Stow the local data area.

Creating a Parameter Data Area from an Existing Local Data Area

A parameter data area (PDA) is used to specify the data parameters to be passed between your Natural program and the subprogram that you will create later. The parameter data area will be referenced in the subprogram.

With minor modifications, your local data area can be used to create the parameter data area: you will delete two of the data fields in in the local data area and then save the revised data area as a parameter data area. The original local data area remains intact.

To create the parameter data area

1 In the local data area, delete the fields `#NAME-START` and `#NAME-END` using the line command `D`.

2 Enter the following in the command line of the data area editor.

```
SA PDA01
```

The current data area is saved with the new name `PDA01`. The existing local data area is still shown in the editor.

3 Load the newly created data area into the editor by entering the following command:

```
E PDA01
```

4 To change the local data area into a parameter data area, enter the following command:

```
SET TYPE A
```

where "A" denotes parameter data area.

The object type changes to "Parameter". This is indicated at the top of the screen. The parameter data area should now look as follows:

```
Command:
 Mem: PDA01    Lib: TUTORIAL Type: PARAMETER  Bytes:   116  Line:      of:   3
C T    Comment
  *    *** Top of Data Area ***
     1 #PERS-ID                          A          8
     1 #MAKE                             A         20
     1 #MODEL                            A         20
  *    *** End of Data Area ***
```

```
  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

5 Stow the parameter data area.

Creating Another Local Data Area Containing a Different View

You will now create a second local data area and import fields from the DDM for the `VEHICLES` database file.

This local data area will be referenced in the subprogram.

To create the local data area

1 Enter the following command in the command line of the data area editor.

```
CLEAR
```

The data area editor is now empty.

2 To change the type of the data area, enter the following in the command line:

```
SET TYPE L
```

where "L" denotes local data area.

3 Press ESC to switch to edit mode. Then enter the following line command in the first column of the line indicating the top of the data area.

```
V
```

4 Enter the following in the resulting window and press ENTER:

```
+------------------- View Definition -------------------+
¦Name of View:          VEHICLES-VIEW                   ¦
¦Name of DDM:           VEHICLES                        ¦
¦Comment:                                               ¦
+-------------------------------------------------------+
```

A window appears showing the fields of the specified DDM.

```
                      Press <ESC> to enter command mode
 Mem: empty    Lib: TUTORIAL Type: LOCAL      Bytes:     0  Line:    0 of:   0
C T    Comment
V *    *** Top of Data Area ***
  *    *** End of Data Area ***









+------------------------------- DDM: VEHICLES ---------------------------------+
¦    1  AA  REG-NUM                          A         15  N D                 ¦
¦    1  AB  CHASSIS-NUM                      B          4  F                   ¦
¦    1  AC  PERSONNEL-ID                     A          8    D                 ¦
¦  G 1  CD  CAR-DETAILS                                                        ¦
¦    2  AD  MAKE                             A         20  N D                 ¦
¦    2  AE  MODEL                            A         20  N                   ¦
+-------------------------------------------------------------------------------+



  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

5 Mark the following fields by entering an "X" in the first column:

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

6 After you have marked all required fields, press ENTER.

The local data area should now look as follows (use the arrow keys to scroll to the top of the data area):

```
                      Press <ESC> to enter command mode
 Mem:          Lib: TUTORIAL Type: LOCAL      Bytes:   485  Line:    0 of:   5
C T    Comment
  *    *** Top of Data Area ***
  V  1 VEHICLES-VIEW                                 VEHICLES
     2 PERSONNEL-ID                     A          8
  G  2 CAR-DETAILS
     3 MAKE                             A         20
     3 MODEL                            A         20
  *    *** End of Data Area ***
```

```
  F 1 HELP     F 2 CHOICE   F 3 QUIT     F 4 SAVE     F 5 STOW     F 6 CHECK
  F 7 READ     F 8 CLEAR    F 9 MEM TYPE F10 GEN      F11 FLD TYPE F12
```

7 Save the new local data area by entering the following in the command line:

```
SA LDA02
```

8 Stow the new local data area.

Creating a Subprogram

You will now create a subprogram that uses a parameter data area and a local data area to retrieve information from the `VEHICLES` file. The subprogram receives the personnel ID passed by the program `PGM01` and uses this ID as the basis for a search of the `VEHICLES` file.

To create the subprogram

1 In the command line of the data area editor, enter the following command:

```
E N
```

where "N" denotes subprogram.

An empty program editor is invoked. The object type has been set to subprogram.

2 Enter the following:

```
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
```

This subprogram returns to a given personnel ID the make and model of the employee's company car.

The `FIND` statement selects a set of records (here: one record) from the database based on the search criterion `#PERS-ID`.

In the field `#PERS-ID`, the subprogram receives the value of `PERSONNEL-ID` that has been passed by the program `PGM01`. The subprogram uses this value as the basis for a search of the `VEHICLES` file.

3 Stow the subprogram.

```
STOW SPGM01
```

Referencing the Subprogram from Your Program

A subprogram is invoked from the main program using a `CALLNAT` statement. A subprogram can only be invoked via a `CALLNAT` statement; it cannot be executed by itself. A subprogram has no access to the global data area used by the invoking object.

Data is passed from the main program to the specified subprogram through a set of parameters that are referenced in the `DEFINE DATA PARAMETER` statement of the subprogram.

The variables defined in the parameter data area of the subprogram do not have to have the same names as the variables in the `CALLNAT` statement. Since the parameters are passed by address, it is only necessary that they match in sequence, format, and length.

You will now modify your main program so that it can use the subprogram you have just defined.

To use the subprogram in your main program

1 Return to the program editor by entering the following in the command line.

```
E PGM01
```

2 Insert the following directly above the DISPLAY statement:

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

The RESET statement sets the values of #MAKE and #MODEL to null values.

3 Delete the line containing the DISPLAY statement and replace it with the following:

```
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
```

The text defined with the WRITE TITLE statement will appear at the top of each page in the output. The WRITE TITLE statement overrides the default page title: the information which was previously displayed at the top of each page (page number, date and time) is no longer shown. Each slash (/) causes the subsequent information to be shown in a new line.

Since the subprogram is now returning additional vehicles information, the columns in the output need to be resized. They receive shorter headers. The column in which the asterisk is to be shown (#MARK), does not receive a header at all. One space will be inserted between the columns (1X). Each slash in the header causes the subsequent information to be shown in a new line of the same column.

Your program should now look as follows:

```
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 Run the program.

5 Enter "JONES" as the starting name and press ENTER.

The resulting list should look similar to the following:

```
MORE

                *** 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 To return to the program editor, enter `EDIT` at the `MORE` prompt.

7 Stow the program.

You have successfully completed this tutorial.