Natural

First Steps

Version 9.2.5

October 2026

ADABAS & NATURAL

This document applies to Natural Version 9.2.5 and all subsequent releases.

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

Copyright © 1979-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software GmbH product names are either trademarks or registered trademarks of Software GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at https://softwareag.com/licenses.

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses 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, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software GmbH Products / Copyright and Trademark Notices of Software GmbH Products". These documents are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software GmbH.

Document ID: NATMF-NNATFIRSTSTEPS-925-20261009

Table of Contents

Preface v
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
2 About this Tutorial 5
Prerequisites 6
About the Sample Application 6
3 Getting Started with Natural 9
Invoking Natural's Main Menu 10
Libraries 11
Issuing Commands 11
Creating a User Library 11
Development Functions Menu 12
Programming Modes 13
4 Hello World! 15
Creating a Program 16
Running a Program 17
Correcting Program Errors 18
Stowing a Program 19
Displaying Information about a Program 20
Displaying the Content of the Current Library 21
Setting the Editor Profile Options 22
5 Database Access 27
Saving Your Program Under a New Name 28
Defining the Required Data Using a View 29
Reading Data from a Database 31
Reading Selected Data from a Database 33
6 User Input 35
Allowing for User Input 36
Designing a Map for User Input 38
Invoking the Map from Your Program 52
Ensuring that an Ending Name is Always Used 54
7 Loops and Labels 57
Allowing Repeated Usage 58
Displaying a Message Indicating that Information was not Found 60
8 Inline Subroutines 63
Defining the Inline Subroutine 64
Performing the Inline Subroutine 65
9 Processing Rules and Helproutines 67
Defining a Processing Rule 68
Defining a Helproutine 70
10 Local Data Areas 73

Creating a Local Data Area 74
Defining Data Fields 75
Importing the Required Data Fields from a DDM 76
Referencing the Local Data Area from Your Program 79
11 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
12 External Subroutines 89
Creating an External Subroutine 90
Referencing the External Subroutine from Your Program 91
13 Subprograms 95
Modifying the Local Data Area 96
Creating a Parameter Data Area from an Existing Local Data Area 97
Creating Another Local Data Area Containing a Different View 99
Creating a Subprogram 101
Referencing the Subprogram from Your Program 102
Index 107

Preface

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 Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 About this Tutorial

- Prerequisites 6
- About the Sample Application 6

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

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 Natural program, data area and map editor have been disabled in your environment by default. For more information, see *NaturalONE as the Default Development Environment* in the *Editors* documentation.

Important: 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.

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.

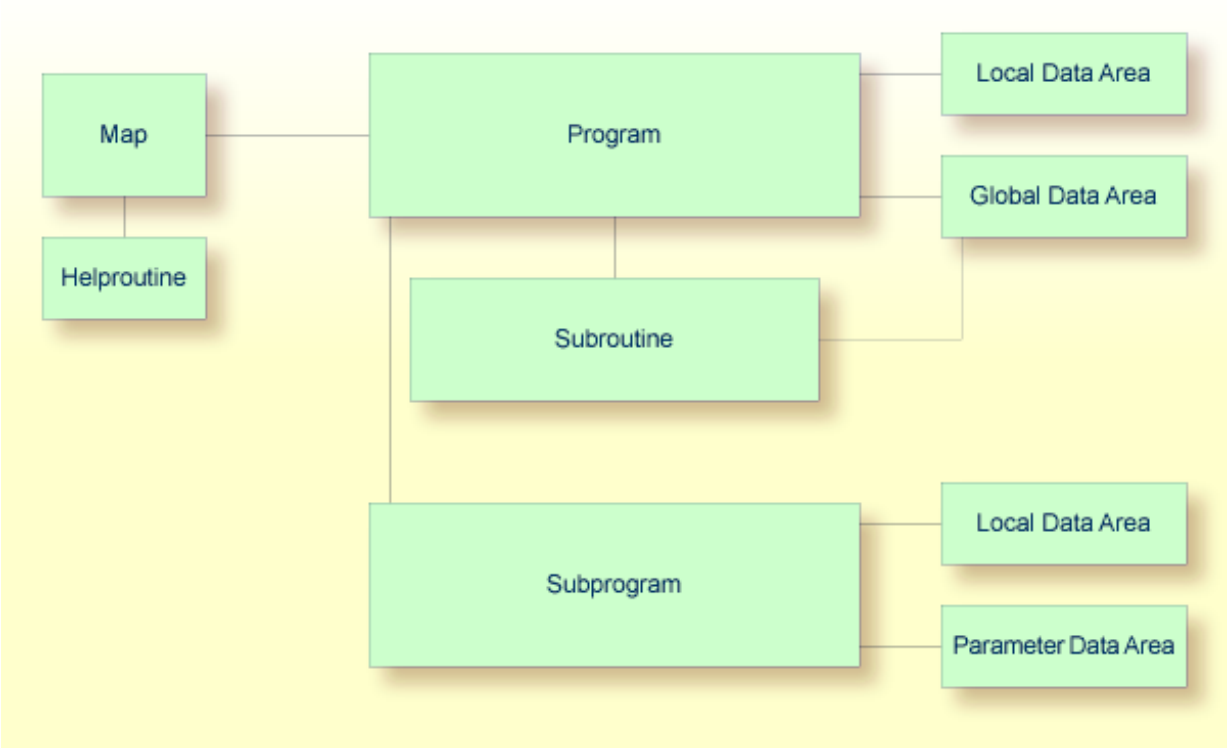
Prerequisites

To perform all steps of this tutorial, the Adabas demo files `EMPLOYEES` and `VEHICLES` must be installed as well as the Natural example objects. If they are not installed, ask your administrator to install them.

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:



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

3 Getting Started with Natural

- Invoking Natural's Main Menu 10
- Libraries 11
- Issuing Commands 11
- Creating a User Library 11
- Development Functions Menu 12
- Programming Modes 13

Invoking Natural's Main Menu

The main menu of Natural provides access to Natural development functions, environment settings, utilities and example libraries.

To invoke Natural's main menu

1 Start Natural according to the procedures at your site.

Depending on the default settings in your environment, either the Natural main menu or the `NEXT` or `MORE` command prompt appears.

2 If one of the above command prompts appears, enter the following:

```
MAINMENU
```

The main menu appears.

```
09:51:48                    *****  NATURAL  *****                    2012-07-17
User SAG                        - Main Menu -                Library SYSTEM


                   Function

                _  Development Functions
                _  Development Environment Settings
                _  Maintenance and Transfer Utilities
                _  Debugging and Monitoring Utilities
                _  Example Libraries
                _  Other Product
                _  Help
                _  Exit Natural Session









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

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 internal 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 (and of many other screens) shows the name of the library where you are currently logged on.

Issuing Commands

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.

To create a user library

- In the command line (indicated by **Command ===>** in the Natural main menu), enter the following:

  ```
  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.

  Or:

In the upper right-hand corner of the screen, overwrite the name of the current library with the name of the library to which you want to log on and press ENTER.

Example:

```
11:33:26                        *****  NATURAL  *****                    2012-07-17
User SAG                            - Main Menu -                Library TUTORIAL
```

Development Functions Menu

The **Development Functions** menu can be used to create and modify Natural objects.

To invoke the Development Functions menu

- On the Natural main menu, enter any character in the input field next to **Development Functions** and press ENTER.

  Or:

  Use cursor selection, that is: place the cursor in the input field next to **Development Functions** and press ENTER.

  The **Development Functions** menu appears.

```
11:56:21                    *****  NATURAL  *****                    2012-07-17
User SAG                  - Development Functions -          Library TUTORIAL
                                                                Mode Structured
                                                           Work area empty

            Code  Function            Code  Function

             C    Create Object        L    List Objects or Single Source
             E    Edit Object          O    List Source with Expanded Sources
             X    Execute Program      N    List Extended Object Names
             R    Rename Object        I    List Directory Information
             D    Delete Objects       U    List Used Subroutines, etc.
             S    Scan Objects         ?    Help
                                       .    Exit


     Code .. _    Type .. _   Name .. ________________________________



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

Programming Modes

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

- Enter the following system command in the command line and press ENTER:

```
GLOBALS SM=ON
```

Or:

In the upper right-hand corner of the **Development Functions** menu, overwrite the first position of the **Mode** field (which shows "Reporting") with the following letter and press ENTER:

```
S
```

Example:

```
12:17:20                    *****  NATURAL  *****                    2012-07-17
User SAG                  - Development Functions -          Library TUTORIAL
                                                                Mode Seporting
                                                            Work area empty
```

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

4 Hello World!

- Creating a Program 16
- Running a Program 17
- Correcting Program Errors 18
- Stowing a Program 19
- Displaying Information about a Program 20
- Displaying the Content of the Current Library 21
- Setting the Editor Profile Options 22

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 At the bottom of the **Development Functions** menu, enter the following information and press ENTER:

```
     Code .. C    Type .. P   Name .. HELLO_____________________________



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

"C" stands for the function **Create Object**, "P" stands for the object type program, and "HELLO" is the name of the program to be created.

Tip: When you enter the function code `C`, you can also enter an asterisk (*) in the **Type** field. When you press ENTER, a list of all object types and the letters that correspond to these object types is shown.

The program editor appears. It is currently empty.

3 Enter the following code in the program editor:

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

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, z/OS or Linux and Cloud). With the z/OS 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
```

System commands may be abbreviated. `R` is the abbreviated form of `RUN`.

Depending on the definitions in your environment, the command line is located either at the top or bottom of the screen.

```
> RUN                                   > +  Program     HELLO    Lib TUTORIAL
```

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

```
MORE
Page      1                                                  13-05-16  13:27:42


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.

```
NAT0305 Text string must begin and end on the same line.
>                                       > +  Program     HELLO    Lib TUTORIAL
All    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 * The "Hello world!" example in Natural.
  0020 *
E 0030 DISPLAY "HELLO WORLD!
  0040 END /* End of program
  0050
  0060
  0070
  0080
  0090
  0100
  0110
  0120
  0130
  0140
  0150
  0160
  0170
  0180
  0190
  0200
       ....+....1....+....2....+....3....+....4....+....5....+... S 4    L 1
```

The statement line that contains the error is highlighted and marked with an "E".

3 Correct the error, that is: insert the missing quotation mark at the end of the line.

4 Run the program once more 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`.

To stow a program

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

```
STOW
```

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.

```
13:15:45               ***** NATURAL LIST COMMAND *****              2012-07-17
User SAG                       - List Directory -               Library TUTORIAL

Directory of Program HELLO                     Saved on ... 2012-07-17 13:15:36
--------------------------------------------------------------------------------
Library .... TUTORIAL   User-ID ...... SAG       Mode ....... Structured
TP-System .. COMPLETE   Terminal-ID .. DAEFTCA9
Op-System .. MVS/ESA    Transaction .. NATvr
NAT-Ver .... v.r.s
Source size ........................     100 Bytes

Directory of Program HELLO                     Cataloged on 2012-07-17 13:15:36
--------------------------------------------------------------------------------
Library .... TUTORIAL   User-ID ...... SAG       Mode ....... Structured
TP-System .. COMPLETE   Terminal-ID .. DAEFTCA9
Op-System .. MVS/ESA    Transaction .. NATvr
NAT-Ver .... v.r.s
Used GDA ...
Size of global data ...       0 Bytes  Size in DATSIZE ......     560 Bytes
Size in buffer pool ...    2620 Bytes

Size of OPT-Code ......       0 Bytes
Initial OPT string ....
                                                              ENTER to continue
```

Note: In the example above, the notations `vr` and `v.r.s` stand for the current version number of Natural. See also the definition of *Version* in the *Glossary*.

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 screen appears. It lists the program you have just created.

```
13:34:27               ***** NATURAL LIST COMMAND *****             2012-07-17
User SAG                 - LIST Objects in a Library -         Library TUTORIAL

Cmd  Name        Type         S/C  SM Version  User ID    Date         Time
---  *________   *__________  *__  *  *______  *________  *__________  *________
__   HELLO       Program      S/C  S  v.r.s    SAG        2012-07-17   13:15:36















                                                               1 Objects found
Top of List.
Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Print Exit  Sort        --    -     +     ++          >     Canc
```

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

The following window appears.

```
+-------- COMMANDS ---------+
!                           !
!   ED   Edit               !
!   LI   List               !
!   LD   List Dir           !
!   PR   Print              !
!   LE   List expanded      !
!   RU   Run                !
!   ST   Stow               !
!   CA   Catalog            !
!   DE   Delete             !
!   RE   Rename             !
!   .    End                !
!                           !
!                           !
!                           !
!                           !
!                           !
!   __ HELLO                !
+---------------------------+
```

3 Do not apply any changes right now. Press PF3 to close the window without specifying any command.

4 Press PF3 once more to return to the program editor.

Setting the Editor Profile Options

When working with the Natural program editor or data area 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 following screen appears.

```
13:35:43                ***** NATURAL EDITORS *****                2012-07-17
                            - Editor Profile -

Profile Name .. SYSTEM__

PF and PA Keys
  PF1 ... HELP___________   PF2 ... ______________   PF3 ... EXIT__________
  PF4 ... ______________    PF5 ... ______________   PF6 ... ______________
  PF7 ... -_____________    PF8 ... +_____________   PF9 ... ______________
  PF10 .. SC=___________    PF11 .. ______________   PF12 .. CANCEL________
  PF13 .. ______________    PF14 .. ______________   PF15 .. MENU__________
  PF16 .. ______________    PF17 .. ______________   PF18 .. ______________
  PF19 .. --____________    PF20 .. ++____________   PF21 .. ______________
  PF22 .. ______________    PF23 .. ______________   PF24 .. ______________
  PA1 ... ______________    PA2 ... SCAN__________   PA3 ... ______________

Automatic Functions
  Auto Renumber .. Y   Auto Save Numbers .. 0__   Source Save into .. EDITWORK

Additional Options .. N

Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit  AddOp Save  Reset                         Del   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 In the **Additional Options** field, enter "Y" and press ENTER.

Or:

Press PF4.

The following window appears.

```
+--------------------- Additional Options ----------------------+
!                                                                !
!                                                                !
!   +  Editor Defaults ......... N                               !
!   +  General Defaults ........ N                               !
!   +  Color Definitions ....... N                               !
!                                                                !
!                                                                !
!                                                                !
!                                                                !
!                                                                !
+----------------------------------------------------------------+
```

3 Enter "Y" in the fields **Editor Defaults** and **General Defaults**, and press ENTER.

The following window appears for the editor defaults.

```
+----------------------- Editor Defaults -----------------------+
|                                                                |
|    Escape Character for Line Command .. .                      |
|    Empty Line Suppression ............. Y                      |
|    Empty Line Suppression for Text .... N                      |
|    Source Size Information ............ N                      |
|    Source Status Message .............. Y                      |
|    Absolute Mode for SCAN/CHANGE ...... N                      |
|    Range Mode for SCAN/CHANGE ......... N                      |
|    Direction Indicator ............... +                       |
|                                                                |
+----------------------------------------------------------------+
```

You can see the escape character that has been defined for line commands. This tutorial assumes that the default character, which is the a period (.), is used.

This tutorial also assumes that the option **Empty Line Suppression** is set to "Y". In this case, all blank lines in the program editor are automatically deleted when you press ENTER. They are not deleted when this option is set to "N".

4 For this tutorial, you should make sure that all options are set as shown above. Press ENTER to display the next window.

The following window appears.

```
+----------------------- General Defaults ----------------------+
!                                                                !
!    Editing in Lower Case ............. N                      !
!    Dynamic Conversion of Lower Case ... Y                      !
!    Position of Message Line ........... TOP                    !
!    Cursor Position in Command Line .... N                      !
!    Stay on Current Screen ............ N                      !
!    Prompt Window for Exit Function .... Y                      !
!    ISPF Editor as Program Editor ...... N                      !
!    Leave Editor with Unlock ........... N                      !
!                                                                !
+----------------------------------------------------------------+
```

When the option **Editing in Lower Case** is set to "Y" and the option **Dynamic Conversion of Lower Case** is set to "N", any source code remains as you enter it. This feature, however, also depends on system-environment-specific settings which may force an uppercase translation of all of your input; this cannot be influenced by Natural.

5 If desired, change the above mentioned options for lowercase conversion and press ENTER. Press ENTER once more to return to the **Additional Options** window, and then press ENTER again to close this window.

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

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

7 Press PF5 to save your changes in the database and then press PF3 to exit the editor profile.

Note: Instead of pressing a PF key, you can also enter the corresponding command in the command line. For example, in the above case, you can enter the commands `SAVE` and `EXIT`.

Or:

If you do not want to save your changes to the database but want to use them for the current session, press PF3 to exit the editor profile.

The exit function displays a window with different options.

```
+----------- EXIT Function -----------+
!                                     !
!   _ Save and Exit                   !
!   _ Exit without Saving             !
!   _ Resume Function                 !
!                                     !
!                                     !
+-------------------------------------+
```

Select the option **Exit without Saving** to use the changes for the current session only.

Or:

If you want to exit the editor profile without keeping any changes, press PF12.

Your program is shown again. Any new settings will now be used in the program editor (and also in the data area editor which is explained later).

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

5 Database Access

- Saving Your Program Under a New Name 28
- Defining the Required Data Using a View 29
- Reading Data from a Database 31
- Reading Selected Data from a Database 33

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:

```
>                                        > +  Program     PGM01    Lib TUTORIAL
All    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 .DThe "Hello world!" example in Natural.
  0020 .D
  0030 .DSPLAY "Hello world!"
  0040 .DD /* End of program
  0050
```

Or:

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

```
.D(4)
```

where the number in parentheses indicates the number of lines to be deleted.

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.

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.

```
>                                     > +  Program     PGM01    Lib TUTORIAL
All    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 DEFINE DATA
  0020 LOCAL
  0040 END-DEFINE
  0050 *
  0060 END
  0070
  0080
  0090
  0100
  0110
       ....+....1....+....2....+....3....+....4....+....5....+... S 5    L 1
Split Top  View EMPLOYEES                   DBID     0 FNR     1 Def seq
    1  AA  PERSONNEL-ID                    A    8    D CNNNNNNN
  G 1  AB  FULL-NAME                                   NAME INFORMATION
    2  AC  FIRST-NAME                      A   20  N   FIRST/CHRISTIAN NA
    2  AD  MIDDLE-I                        A    1  N   MIDDLE INITIAL
    2  AE  NAME                            A   20    D SURNAME/FAMILY NAM
    1  AD  MIDDLE-NAME                     A   20  N   SECOND/MIDDLE NAME
    1  AF  MAR-STAT                        A    1  F   M=MARRIED
    1  AG  SEX                             A    1  F
    1  AH  BIRTH                            D    6  N D BIRTH-DATE (YYYY-M
```

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:

Command	Description
SPLIT + or S +	Page forward in the view.
SPLIT - or S -	Page backward in the view.
SPLIT . or S .	Terminate split-screen mode.

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:

```
.I
```

In full-screen mode, 9 blank lines are inserted. Only 4 blank lines would have been inserted in split-screen mode.

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.

Note: The remaining blank lines are not eliminated, when the default setting in the editor profile has been changed, that is: when the option **Empty Line Suppression** has been set to "N".

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 last transaction has been backed out of the database. 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 or ENTER	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.

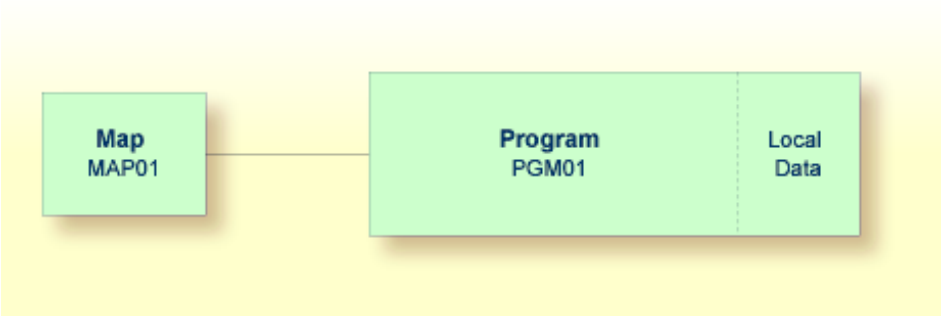
You can now proceed with the next exercises: ***User Input***.

6 User Input

- Allowing for User Input 36
- Designing a Map for User Input 38
- Invoking the Map from Your Program 52
- Ensuring that an Ending Name is Always Used 54

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:

```
Ob _                                    Ob D CLS ATT  DEL    CLS ATT  DEL
.                                       .     T  D    Blnk    T  I    ?
.                                       .     A  D    _       A  I    )
.                                       .     A  N    ^       M  D    &
.                                       .     M  I    :       O  D    +
.                                       .     O  I    (
.                                       .
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
(XXXXXXXXXX                                                        (XXXXXXXX


                          Start :XXXXXXXXXXXXXXXXXXXX

                            End :XXXXXXXXXXXXXXXXXXXX









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Mset  Exit  Test  Edit  --    -     +     Full  <     >     Let
```

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 length of a field is indicated by a number of "X" characters. Delimiters are used to distinguish the different types of fields. In our sample map, the following default delimiters are used:

Delimiter	Type of field
(	System variable.
:	Data field.

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. The map editor can be accessed using the **Edit Map** menu.

To access the Edit Map menu

1. Enter a dot (.) in the command line of the program editor to return to the **Development Functions** menu.
2. At the bottom of the **Development Functions** menu, enter the following information and press ENTER:

```
     Code .. E    Type .. M   Name .. ________________________________



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

"E" stands for the function **Edit Object**, and "M" stands for the object type map.

Note: It would also have been possible to enter the code "C" for **Create Object**. But in this case, you would have been prompted to enter an object name.

The **Edit Map** menu appears.

```
16:43:41                ***** NATURAL MAP EDITOR *****              2012-07-17
User SAG                         - Edit Map -                  Library TUTORIAL

                   Code    Function
                   ----    --------------------------------
                     D     Field and Variable Definitions
                     E     Edit Map
                     I     Initialize new Map
                     H     Initialize a new Help Map
                     M     Maintenance of Profiles & Devices
                     S     Save Map
                     T     Test Map
                     W     Stow Map
                     ?     Help
                     .     Exit



             Code .. I     Name .. ________       Profile .. SYSPROF_

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

To initialize a new map

1 At the bottom of the **Edit Map** menu, enter the following information and press ENTER:

```
             Code .. I     Name .. MAP01___       Profile .. SYSPROF_

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

The **Define Map Settings** screen appears.

```
16:43:42                Define Map Settings for MAP                  2012-07-17

 Delimiters           Format                          Context
 -----------------    ---------------------------    --------------------------
 Cls Att CD  Del      Page Size ...... 23            Device Check .... ________
  T   D      BLANK    Line Size ...... 79            WRITE Statement   _
  T   I      ?        Column Shift ... 0 (0/1)       INPUT Statement   X
  A   D      _        Layout ......... ________      Help  ____________________
  A   I      )         dynamic ....... N (Y/N)        as field default N (Y/N)
  A   N      ^        Zero Print ..... N (Y/N)
  M   D      &        Case Default ... UC (UC/LC)
  M   I      :        Manual Skip .... N (Y/N)       Automatic Rule Rank  1
  O   D      +        Decimal Char ... .             Profile Name .... SYSPROF
  O   I      (        Standard Keys .. N (Y/N)
                      Justification .. L (L/R)       Filler Characters
                      Print Mode ..... __            ------------------------
                                                     Optional, Partial ....
                      Control Var .... ________      Required, Partial ....
                                                     Optional, Complete ...
                                                     Required, Complete ...

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                  Let
```

In this screen, you define the default settings for a map (for example, the size of the map). It is possible to change the delimiter characters in this screen. However, for this tutorial, you will use the default delimiter characters. You will only define the filler characters (see the next step).

The delimiter character indicates the combination of class and attribute assigned to a field. The delimiters indicated in bold are used in this tutorial:

```
Delimiters
 -----------------
 Cls Att CD  Del
  T   D      BLANK
  T   I      ?
  A   D      _
  A   I      )
  A   N      ^
  M   D      &
  M   I      :
  O   D      +
  O   I      (
```

- The colon identifies a field as a modifiable input and output field ("M" in the **Cls** column) and intensified ("I" in the **Att** column).
- The opening parenthesis identifies the field as output only ("O" in the **Cls** column) and intensified ("I" in the **Att** column).

- When you do not use a delimiter character (indicated with "BLANK" in the screen), the field is identified as a text constant ("T" in the **Cls** column) with default attibutes ("D" in the **Att** column).

2 Enter an underscore (_) for each of the filler character options as shown below:

```
Filler Characters
 ------------------------
 Optional, Partial .... _
 Required, Partial .... _
 Optional, Complete ... _
 Required, Complete ... _
```

A 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.

3 Press ENTER to save the changes.

4 Press ENTER once more to invoke the map editing area.

The map editing area is shown in split-screen mode.

```
Ob _                                    Ob D CLS ATT  DEL    CLS ATT  DEL
.                                       .     T  D    Blnk    T  I    ?
.                                       .     A  D    _       A  I    )
.                                       .     A  N    ^       M  D    &
.                                       .     M  I    :       O  D    +
.                                       .     O  I    (
.                                       .
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Mset  Exit  Test  Edit  --    -     +     Full  <     >     Let
```

Note: The **Define Map Settings** screen only appears when you initialize a new map. When you edit a map, the **Define Map Settings** screen is invoked if you press PF2 (Mset) in the map editing area (see the above screen).

In split-screen mode, the upper half of the screen shows the valid delimiter characters. The lower half of the screen is the editing area in which you will create the map.

Unlike the program editor and the data area editor (described later in this tutorial), the map editor does not have a command line or command prompt where you can enter Natural system commands. Many functions in the map editor are performed by using line or field commands (see below) or by using PF keys.

You can press PF9 to toggle between split-screen mode and full-screen mode, which displays the editing area in full size.

Defining Text Fields

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

To define the text fields

1 In the map editing area, move the cursor to the first position of the fourth line and type in the following:

```
Start
```

2 Move the cursor to the first position of the next line and type in the following:

```
End
```

Your map should now look as follows:

```
Ob _                                   Ob D CLS ATT  DEL    CLS ATT  DEL
.                                      .     T  D    Blnk    T  I    ?
.                                      .     A  D    _       A  I    )
.                                      .     A  N    ^       M  D    &
.                                      .     M  I    :       O  D    +
.                                      .     O  I    (
.                                      .
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----



Start
End



Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Mset  Exit  Test  Edit  --    -     +     Full  <     >     Let
```

Note: If you want to cancel your last changes, press PF12 before you press ENTER.

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.

You can define the data fields in two different ways: in the classic way where it is your responsibility to define the correct length of the data field, or in a more user-friendly way where you simply select the data field from a list and where the correct length is automatically used. These two ways are described below.

To define a data field where you have to specify the length

1 Type in the following behind the **Start** text field (leave a blank space between text field and data field):

```
:X(20)
```

The colon (:) is the delimiter character which indicates that the data field is modifiable and intensified. The data field is defined with a length of 20 characters. The length of the field is indicated by "X" characters.

2 Press ENTER.

Your map should now look as follows:

```
Ob _                                    Ob D CLS ATT  DEL    CLS ATT  DEL
.                                       .     T  D    Blnk    T  I    ?
.                                       .     A  D    _       A  I    )
.                                       .     A  N    ^       M  D    &
.                                       .     M  I    :       O  D    +
.                                       .     O  I    (
.                                       .
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----



Start :XXXXXXXXXXXXXXXXXXXX
End









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Mset  Exit  Test  Edit  --    -     +     Full  <     >     Let
```

To select a data field from a list

1 In the **Ob** field, which is located at the top left of the screen, enter the following and press ENTER:

```
P PGM01
```

The data fields that are currently used by the program `PGM01` are now shown in the screen. The fields that can be used in the map are preceded by a number.

```
Ob P PGM01                                Ob D CLS ATT  DEL    CLS ATT  DEL
1 #NAME-START                A20          .     T  D    Blnk    T  I    ?
2 #NAME-END                  A20          .     A  D    _       A  I    )
. EMPLOYEES-VIEW             *V1          .     A  N    ^       M  D    &
. FULL-NAME                  *2           .     M  I    :       O  D    +
3 NAME                       A20          .     O  I    (
4 DEPT                       A6           .
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----



Start :XXXXXXXXXXXXXXXXXXXX
End




Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Mset  Exit  Test  Edit  --    -     +     Full  <     >     Let
```

Not all data fields defined in PGM01 are shown in the screen. To page through the list of data fields, enter one of the following positioning commands in the **Ob** field (that is: in the field which currently contains the letter "P"):

Command	Description
+	Page forward in the list.
-	Page backward in the list.
++	Go to the bottom of the list.
--	Go to the top of the list.

2 Type in the following behind the **End** text field (leave a blank space between text field and data field) and press ENTER:

```
:2
```

The colon (:) is the delimiter character which indicates that the data field is modifiable and intensified. 2 is the number assigned to #NAME-END.

The data field is automatically defined with the correct length (20 characters in this case). The length of the field is indicated by "X" characters.

Specifying Names for Data Fields

The following applies only for the data field for the starting name which you have defined manually. It does not apply to the data field for the ending name which you have selected from a list: 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 Starting in the first position of the data field for the starting name, enter the following and press ENTER:

```
.E
```

Or:

Position the cursor anywhere in the data field and press PF5.

The extended field editing section is displayed for the specified field.

```
Fld #001                                                        Fmt A20
--------------------------------------------------------------------------------
AD= MIT'_'____      ZP=         SG=        HE= ______________________   Rls 0
AL= _____           CD= __      CV= ________________________________   Mod Undef
PM= __  DF=                     DY= ______________________________
EM= ___________________________ SB= ________________________________

001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----



Start .EXXXXXXXXXXXXXXXXXXX
End :XXXXXXXXXXXXXXXXXXXX









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP  Mset  Exit  <---  --->  --    -     +           <     >     Let
```

The **Fld** field in the upper left corner of the screen shows the field name that has been assigned by Natural: "#001".

2 In the **Fld** field, enter "#NAME-START".

3 Press PF3 to leave the extended field editing section.

Since the data field for the end value has been selected from a list in the previous exercise, it is not required to repeat the above steps for the ending name. `#NAME-END` is already defined in this case. If you want, you can check this with the line command `.E` as described above.

Note: For `#NAME-END`, the session parameter `AD` has the additional value "L" which has not been defined for `#NAME-START`. "L" means that the value of the field is displayed left-justified. Since this is the default value for alphanumeric fields, it is not necessary to define this for `#NAME-START`.

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 Move the cursor to the first position of the first line and type in the following:

```
(*DAT4I
```

The opening parenthesis is the delimiter character which identifies the system variable as output only and intensified.

2 Move the cursor to the first position of the second line and type in the following:

```
(*TIMX
```

Your map should now look as follows:

```
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
(*DAT4I
(*TIMX

Start :XXXXXXXXXXXXXXXXXXXX
End :XXXXXXXXXXXXXXXXXXXX
```

3 Press ENTER.

"X" characters are now shown instead of the system variable names.

Repositioning Fields

You will now use line commands and field commands to reposition the fields you have added.

To move one field

1 Starting in the first position of the system variable in the second line, enter the following field command:

```
.M
```

Example:

```
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
(XXXXXXXXXX
.MXXXXXXX

Start :XXXXXXXXXXXXXXXXXXXX
End :XXXXXXXXXXXXXXXXXXXX
```

A field command is entered at the beginning of a field. It applies only to the field in which you enter it.

2 Move the cursor to the position to which you want to move the system variable (column 70 of the first line).

3 Press ENTER.

The system variable is moved to the cursor position.

To insert a blank line

1 Starting in the first position of the fourth line (starting name), enter the following line command:

```
..I(1)
```

Example:

```
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
(XXXXXXXXXX                                                        (XXXXXXXX


..I(1):XXXXXXXXXXXXXXXXXXXX
End :XXXXXXXXXXXXXXXXXXXX
```

A line command is entered at the beginning of a line. It applies to the whole line in which you enter it.

2 Press ENTER.

A blank line is inserted below the line in which you have entered the line command.

To center a line

1 Starting in the first position of the fourth line (starting name), enter the following line command:

```
..C
```

Example:

```
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
(XXXXXXXXXX                                                        (XXXXXXXX


..Crt :XXXXXXXXXXXXXXXXXXXX

End :XXXXXXXXXXXXXXXXXXXX
```

2 Press ENTER.

The line is centered.

To move more than one field

1 Enter the following field command starting in the first position of the text field in the sixth line (**End**) and in the first position of the data field in this line:

```
.M
```

Example:

```
001   --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
(XXXXXXXXXX                                                       (XXXXXXXX


                          Start :XXXXXXXXXXXXXXXXXXXX

.Md .MXXXXXXXXXXXXXXXXXXX
```

2 Move the cursor to the position to which the text field is to start (column 30 of the sixth line).

3 Press ENTER.

Both fields are moved.

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 PF4.

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 PF3 to return to the **Edit Map** menu.

2 Enter the following in the **Code** field and press ENTER:

```
W
```

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 Return to the program editor by entering one of the following in the command line of the **Edit Map** menu.

```
EDIT PGM01
```

```
E PGM01
```

2 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
```

3 Run the program.

Your map 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.

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.

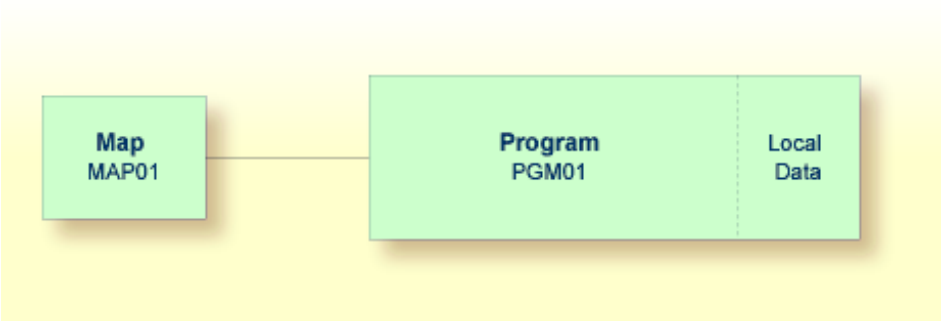
You can now proceed with the next exercises: ***Loops and Labels***.

7 Loops and Labels

- Allowing Repeated Usage 58
- Displaying a Message Indicating that Information was not Found 60

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***.

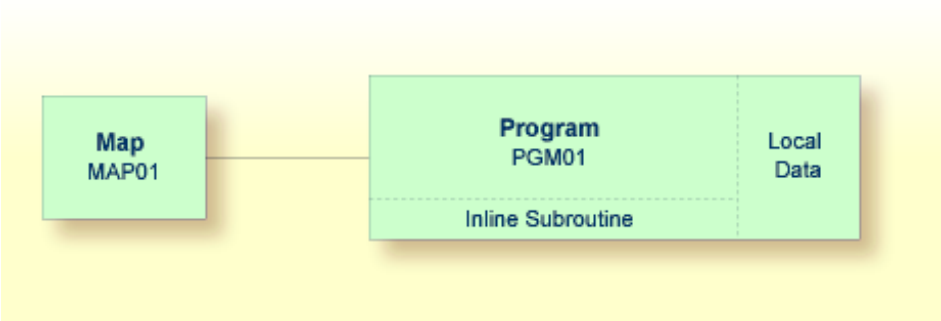
8 Inline Subroutines

- Defining the Inline Subroutine 64
- Performing the Inline Subroutine 65

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.

6 Enter a dot (.) in the command line to return to the **Development Functions** menu.

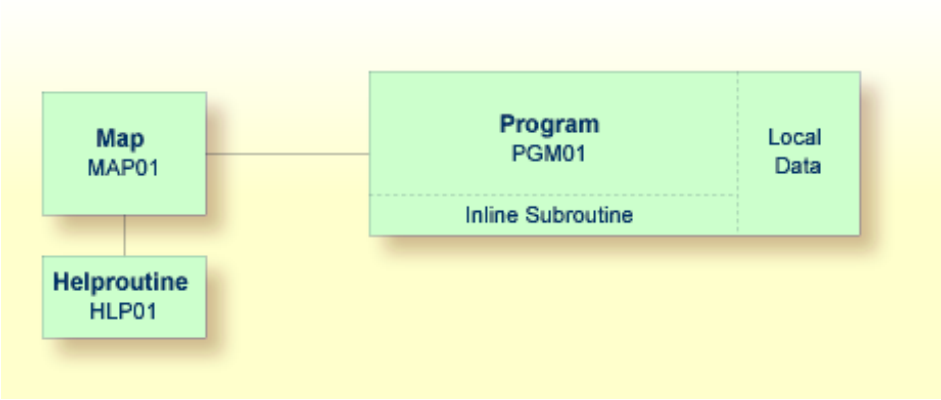
You can now proceed with the next exercises: ***Processing Rules and Helproutines***.

9 Processing Rules and Helproutines

- Defining a Processing Rule 68
- Defining a Helproutine 70

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 in the **Development Functions** menu.

```
     Code .. E    Type .. _   Name .. MAP01_____________________________



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

2 Starting in the first position of the input field for the starting name, enter the following:

```
.P
```

Example:

```
                          Start .PXXXXXXXXXXXXXXXXXX

                            End :XXXXXXXXXXXXXXXXXXX
```

3 Press ENTER

The following screen is displayed for the selected field:

```
Variables used in current map                                            Mod
#NAME-START(A20)                                                           U
#NAME-END(A20)                                                             U





Rule ________________________________ Field #NAME-START
>                                     > + Rank 0    S      L 1     Struct Mode
ALL    ....+....10...+....+....+....30...+....+....+....50...+....+....+....70.
  0010
  0020
  0030
  0040
  0050
  0060
  0070
  0080
  0090
  0100
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Mset  Exit  Test        --    -     +     Full  Sc=         Let
```

4 Enter the following processing rule:

```
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.

5 In the **Rank** field, enter "1".

```
Rule ___________________________________ Field #NAME-START
>                                        > + Rank 1    S 2    L 1     Struct Mode
ALL    ....+....10...+....+....+....30...+....+....+....50...+....+....+....70.
  0010 IF & = ' ' THEN
  0020   REINPUT 'Please enter a starting name.'
  0030   MARK *&
  0040 END-IF
  0050
```

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.

6 Press ENTER to save your input. Then press PF3 to return to the map.

Note: If you want to redisplay your processing rule, you have to use the command `.P1` (to display the rule with the rank 1) or `.P*` (to display a list of all rules defined for this field).

7 Test the map.

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

The output screen is closed.

9 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.

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

11 Stow the map (press PF3 to return to the **Edit Map** menu and enter "W" in the **Code** field).

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 Return to the **Development Functions** menu by pressing PF3 in the **Edit Map** menu.

2 At the bottom of the **Development Functions** menu, enter the following information and press ENTER:

```
     Code .. C    Type .. H   Name .. HLP01____________________________



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

"C" stands for the function **Create Object**, "H" stands for the object type helproutine, and "HLP01" is the name of your new helproutine.

An empty editor appears.

3 Enter the following:

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

4 Stow the helproutine.

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 Starting in the first position of the data field for the starting name, enter the following and press ENTER:

```
.E
```

Or:

Position the cursor anywhere in the data field and press PF5.

The extended field editing section is displayed for the field.

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

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

```
Fld #NAME-START                                                    Fmt A20
-------------------------------------------------------------------------------
AD= MIT'_'____      ZP=         SG=        HE= 'HLP01'_______________   Rls 2
AL= _____           CD= __      CV= ____________________________________   Mod User
PM= __  DF=                     DY= ________________________________
EM= ____________________________ SB= ____________________________________
```

4 Press PF3 to leave the extended field editing section.

5 Test the map.

6 In the resulting output, enter a question mark (?) in the input field for the starting name and press ENTER.

The help text you have defined is shown.

7 Press ENTER to return to the map.

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

9 Stow the map (press PF3 to return to the **Edit Map** menu and enter "W" in the **Code** field).

10 Press PF3 to return to the **Development Functions** menu.

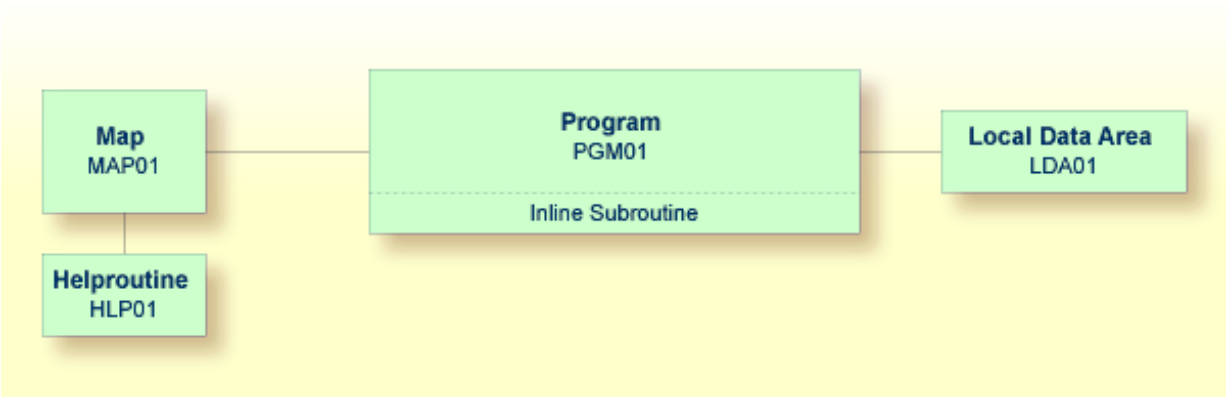
You can now proceed with the next exercises: ***Local Data Areas***.

10 Local Data Areas

- Creating a Local Data Area 74
- Defining Data Fields 75
- Importing the Required Data Fields from a DDM 76
- Referencing the Local Data Area from Your Program 79

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

- At the bottom of the **Development Functions** menu, enter the following information and press ENTER:

```
     Code .. C    Type .. L   Name .. LDA01_____________________________



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

  "C" stands for the function **Create Object**, "L" stands for the object type local data area, and "LDA01" is the name of your new local data area.

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

```
Local     LDA01      Library TUTORIAL                        DBID 11177 FNR    8
Command                                                                      > +
I T L  Name                            F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->




















---------------------------------------------------------------- S 0     L 1
```

Defining Data Fields

You will now define the following fields:

Level (L column)	Name	Format (F column)	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. To display your program and the data area editor on the same screen, enter the following to invoke split-screen mode.

```
SPLIT P PGM01
```

The screen is divided into two sections. Your program is shown in the lower half of the screen. It cannot be modified in this mode. You can use the program as a reference to insert the definitions of the user-defined variables in the data area editor. To page forward and backward in the program, use the commands `SPLIT +` and `SPLIT -`.

2 Specify all required information as listed in the above table.

The local data area should now look as follows:

```
Local    LDA01      Library TUTORIAL                      DBID 11177 FNR     8
Command                                                                    > +
I T L  Name                            F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->
     1 #NAME-START                      A         20
     1 #NAME-END                        A         20
     1 #MARK                            A          1









---------------------------------------------------------------- S 0    L 1
Program     PGM01     Library TUTORIAL
  0010 DEFINE DATA
  0020 LOCAL
  0030   1 #NAME-START        (A20)
  0040   1 #NAME-END          (A20)
  0050   1 #MARK              (A1)
  0060   1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
```

3 Enter the following command to terminate split-screen mode:

```
SPLIT .
```

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).

To import data fields from a DDM

1 In the line below the variables you have already defined, enter the following, starting in the **T** column:

```
.V(EMPLOYEES)
```

Example:

```
Local    LDA01       Library TUTORIAL                      DBID 11177 FNR    ↩
 8
 Command                                                                   > +
 I T L  Name                              F Length     Miscellaneous         ↩

 All -- -------------------------------- - ---------- ------------------------>
      1 #NAME-START                       A         20                        ↩

      1 #NAME-END                         A         20                        ↩

      1 #MARK                             A          1                        ↩

   . V( EMPLOYEES)
```

2 Press ENTER.

The EMPLOYEES view appears.

```
SYSGDA 4461: Mark fields to incorporate into data area.
Local     LDA01     Library TUTORIAL                      DBID 11177 FNR     8
View EMPLOYEES
I T L  Name                              F Length     Miscellaneous
- - -- -------------------------------- - ---------- ------------------------
     2 PERSONNEL-ID                      A          8 /* CNNNNNNN
  G  2 FULL-NAME                                      /* NAME INFORMATION
     3 FIRST-NAME                        A         20 /* FIRST/CHRISTIAN NAME
     3 MIDDLE-I                          A          1 /* MIDDLE INITIAL
     3 NAME                              A         20 /* SURNAME/FAMILY NAME
     2 MIDDLE-NAME                       A         20 /* SECOND/MIDDLE NAME
     2 MAR-STAT                          A          1 /* M=MARRIED
     2 SEX                               A          1
     2 BIRTH                             D            /* BIRTH-DATE (YYYY-MM-
     2 N§BIRTH                           I          2 /* INDICATOR OF BIRTH
  G  2 FULL-ADDRESS
  M  3 ADDRESS-LINE                      A         20 (1:8)/* ALL ADDRESS LINES
     3 CITY                              A         20 /* MAIN CITY/TOWN
     3 ZIP                               A         10 /* POSTAL ADDRESS CODE
     3 POST-CODE                         A         10 /* POSTAL ADDRESS CODE
     3 COUNTRY                           A          3 /* COUNTRY CODE
  G  2 TELEPHONE
- - -- -------------------------------- - ---------- ------------------------
```

3 Mark the following fields by entering any character in the **I** column :

```
PERSONNEL-ID
FULL-NAME
```

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

Not all of these fields are shown on the first page of the view. To scroll forward in the view, press ENTER

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

4 After you have marked all required fields, continue to press ENTER until the data area editor is shown again.

The local data area should now look as follows:

```
SYSGDA 4462: 6 field(s) of view EMPLOYEES included.
Local     LDA01     Library TUTORIAL                      DBID 11177 FNR     8
Command                                                                    > +
I T L  Name                            F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->
     1 #NAME-START                     A         20
     1 #NAME-END                       A         20
     1 #MARK                           A          1
  V  1 EMPLOYEES-VIEW                                EMPLOYEES
     2 PERSONNEL-ID                    A          8 /* CNNNNNNN
  G  2 FULL-NAME                                    /* NAME INFORMATION
     3 NAME                            A         20 /* SURNAME/FAMILY NAME
     2 DEPT                            A          6 /* DDDDSS
  G  2 LEAVE-DATA                                   /* LEAVE/VACATION INFO
     3 LEAVE-DUE                       N        2.0 /* VACATION DAYS/YEAR









---------------------------------------------------------------------- S 10   L 1
```

"-VIEW" has automatically been added to the name of the view. This is the same name that you have already used in your program.

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 Stow the local data area.

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***.

11 Global Data Areas

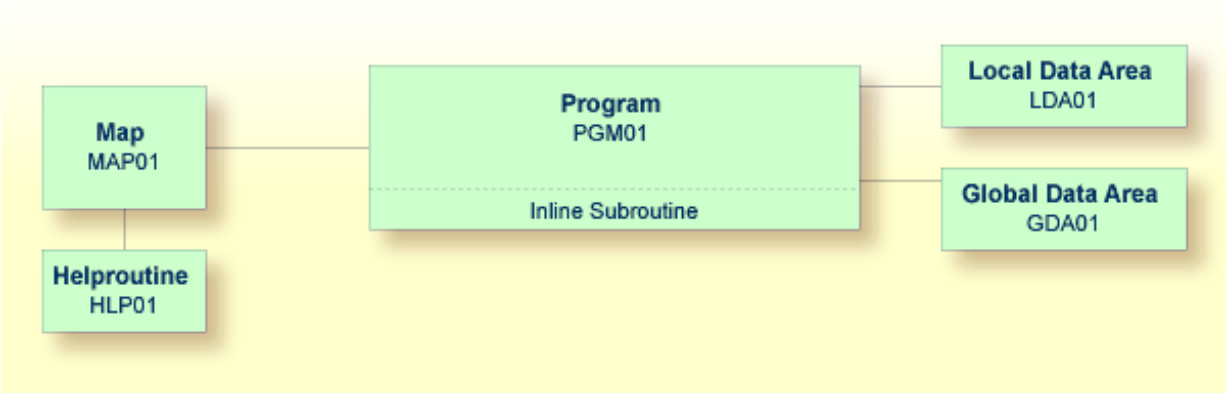
- Creating a Global Data Area from an Existing Local Data Area 82
- Adapting the Local Data Area 84
- Referencing the Global Data Area from Your Program 85

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 left of the screen.

5 Use the line command `.D` to delete the following fields:

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

The line command is entered starting in the **T** column of the line containing the field to be deleted. Since the above fields are defined in two successive lines, you can use the line command `.D(2)` to delete them at the same time.

6 Press ENTER.

The global data area should now look as follows:

```
Global    GDA01     Library TUTORIAL                      DBID 11177 FNR     8
Command                                                                    > +
I T L  Name                            F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->
     1 #MARK                           A          1
  V  1 EMPLOYEES-VIEW                                EMPLOYEES
     2 PERSONNEL-ID                    A          8 /* CNNNNNNN
  G  2 FULL-NAME                                    /* NAME INFORMATION
     3 NAME                            A         20 /* SURNAME/FAMILY NAME
     2 DEPT                            A          6 /* DDDDSS
  G  2 LEAVE-DATA                                   /* LEAVE/VACATION INFO
     3 LEAVE-DUE                       N        2.0 /* VACATION DAYS/YEAR









---------------------------------------------------------------- S 8    L 1
```

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:

```
SYSGDA 4454: Data area stowed successfully.
Local     LDA01     Library TUTORIAL                      DBID 11177 FNR     8
Command                                                                    > +
I T L  Name                            F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->
     1 #NAME-START                     A         20
     1 #NAME-END                       A         20

---------------------------------------------------------------- S 2    L 1
```

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.

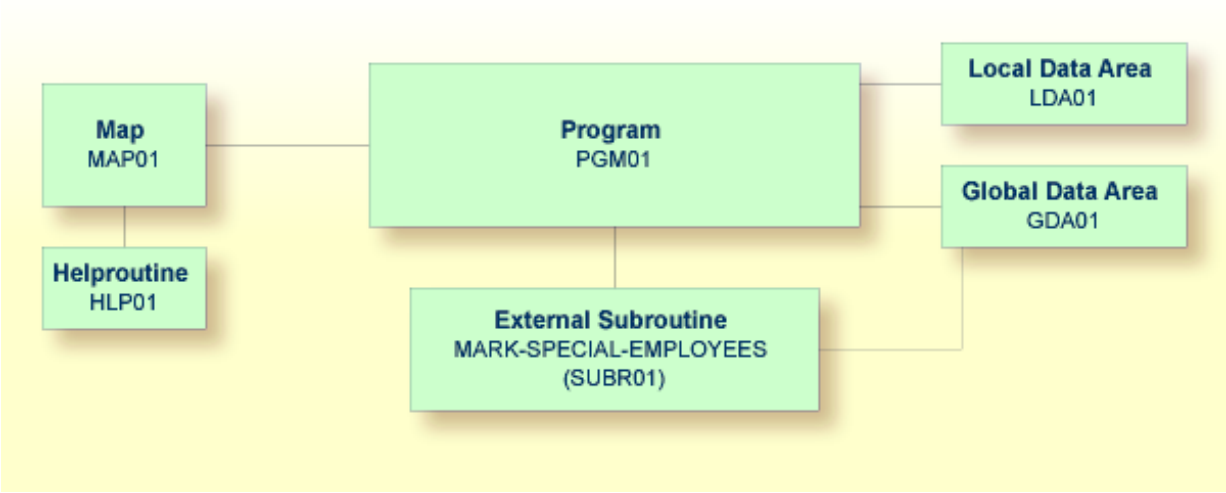
You can now proceed with the next exercises: ***External Subroutines***.

12 External Subroutines

- Creating an External Subroutine 90
- Referencing the External Subroutine from Your Program 91

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.

To list equivalent subroutine names

1 Enter one of the following commands in the command line of the program editor:

```
LIST EXTENDED SUBROUTINE *
```

```
L EXT S *
```

The following screen appears. It lists all external subroutine objects (members) and their equivalent long names available in the current Natural library and system file.

```
12:21:09              ***** NATURAL LIST COMMAND *****              2012-07-17
User SAG                - LIST Objects in a Library -         Library TUTORIAL

Cmd Subroutine/Class Name            Type  S/C Member    Cat Date    Cat Time
--- *________________________________ S____ --- *________ *__________ *________
__  MARK-SPECIAL-EMPLOYEES           Subro S/C SUBR01    2009-06-30  12:11:56

                                                                 1 Objects found
Top of List.
Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Print Exit  Sort        --    -     +     ++          >     Canc
```

If you want to display a certain range of subroutine names, you can enter a search value followed by an asterisk (*) in the field below the header **Subroutine/Class Name** or below the header **Member**. Example:

```
Cmd Subroutine/Class Name            Type  S/C Member    Cat Date    Cat Time
--- MARK*____________________________ S____ --- *________ *__________ *________
__  MARK-SPECIAL-EMPLOYEES            Subro S/C SUBR01    2009-06-30  12:11:56
```

2 Press PF3 to return to the program editor.

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

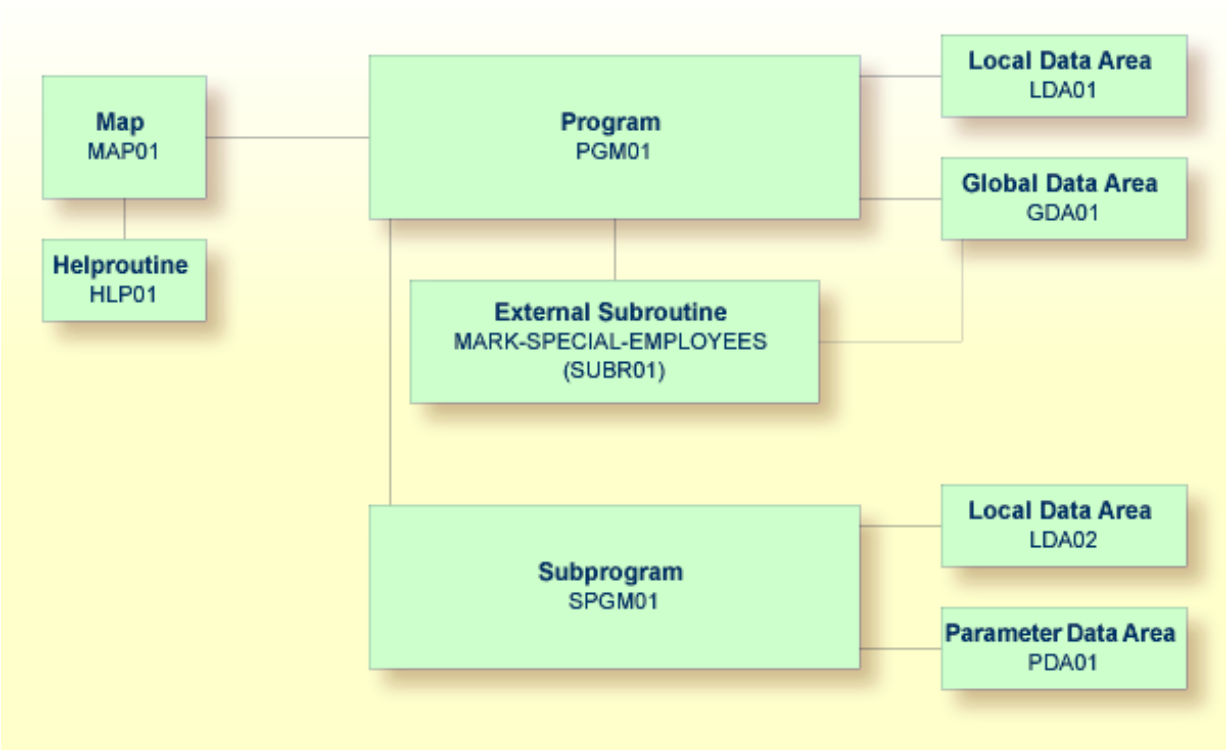
13 Subprograms

- Modifying the Local Data Area 96
- Creating a Parameter Data Area from an Existing Local Data Area 97
- Creating Another Local Data Area Containing a Different View 99
- Creating a Subprogram 101
- Referencing the Subprogram from Your Program 102

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 Define the following fields below #NAME-END:

Level (L column)	Name	Format (F column)	Length
1	#PERS-ID	A	8
1	#MAKE	A	20
1	#MODEL	A	20

The local data area should now look as follows:

```
Local     LDA01     Library TUTORIAL                        DBID 11177 FNR     8
Command                                                                    > +
I T L  Name                              F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->
     1 #NAME-START                       A         20
     1 #NAME-END                         A         20
     1 #PERS-ID                          A          8
     1 #MAKE                             A         20
     1 #MODEL                            A         20

------------------------------------------------------------------ S 5    L 1
```

3 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.

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 left of the screen. The parameter data area should now look as follows:

```
Parameter PDA01     Library TUTORIAL                      DBID 11177 FNR     8
Command                                                                    > +
I T L  Name                            F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->
     1 #PERS-ID                        A          8
     1 #MAKE                           A         20
     1 #MODEL                          A         20
-------------------------------------------------------------------- S 3    L 1
```

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 In the first line of the editing area, enter the following starting in the **T** column:

```
.V(VEHICLES)
```

4 Press ENTER.

The `VEHICLES` view appears.

```
SYSGDA 4461: Mark fields to incorporate into data area.
Local               Library TUTORIAL                      DBID 11177 FNR     8
View VEHICLES
I T L  Name                            F Length     Miscellaneous
- - -- -------------------------------- - ---------- -------------------------
     2 REG-NUM                         A         15 /* CAR'S REGISTR. NUMBE
     2 CHASSIS-NUM                     I          4 /* MANUFACTURER NUMBER
     2 PERSONNEL-ID                    A          8 /* IDENT. OF CAR USER
  G  2 CAR-DETAILS                                   /* DESCRIPTION OF THE C
     3 MAKE                            A         20
     3 MODEL                           A         20
     3 COLOR                           A         10
     3 COLOUR                          A         10
     2 YEAR                            N        4.0 /* MANUFACTURING YEAR
     2 CLASS                           A          1 /* P=PRIVAT
     2 LEASE-PUR                       A          1 /* L=LEASED
     2 DATE-ACQ                        N        8.0 /* DATE THE CAR WAS ACQ
     2 CURR-CODE                       A          3 /* CURRENCY OF CAR COST
  M  2 MAINT-COST                      P        7.0 (1:60)/* MAINTENANCE COST
     2 MODEL-YEAR-MAKE                 A         24 /* YEAR + CAR MAKE /* SP


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

5 Mark the following fields by entering any character in the **I** column :

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

6 After you have marked all required fields, press ENTER to return to the data area editor.

The local data area should now look as follows:

```
Local                Library TUTORIAL                      DBID 11177 FNR     8
Command                                                                      > +
I T L  Name                             F Length     Miscellaneous
All -- -------------------------------- - ---------- ------------------------->
  V  1 VEHICLES-VIEW                                 VEHICLES
     2 PERSONNEL-ID                     A          8 /* IDENT. OF CAR USER
  G  2 CAR-DETAILS                                   /* DESCRIPTION OF THE CAR
     3 MAKE                             A         20
     3 MODEL                            A         20
-------------------------------------------------------------------- S 5    L 1
```

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.

Index

C

command
    issue, 11

E

editor
    invoke, 16
external subroutine
    use with program, 89

G

global data area
    use with program, 81

H

hello world, 15
helproutine
    use in map, 67

I

inline subroutine
    use in program, 63

L

label
    use in program, 57
library
    create, 11
local data area
    use with program, 73
loop
    use in program, 57

M

main menu
    invoke, 10
map
    create, 38
    invoke from program, 52

N

Natural
    invoke main menu, 10

P

processing rule
    use in map, 67
program
    correct error, 18
    create, 16
    run, 17
    save, 28
    stow, 19
programming mode, 13

R

reporting mode, 13

S

structured mode, 13
subprogram
    invoke from program, 95
subroutine
    use with program, 63, 89