

Tutorial - Getting Started With Natural

This tutorial is designed to provide a gradual exposure to specific features of the Natural programming environment and illustrate how applications can be modularized. It is *not* intended to provide an example of how an application should be built.

These sessions also represent a general introduction to how the editors may be used. Therefore explanations are kept to a minimum. For a full description of all editor functions and features, please refer to corresponding sections later in this documentation. This tutorial is *not* intended to be a comprehensive description of the full range of possibilities provided by the Natural editors.

The tutorial consists of the following steps:

- Selecting a Function or Item
- Session 1 - Creating a Program and a Map
- Session 2 - Creating a Local Data Area
- Session 3 - Creating a Global Data Area
- Session 4 - Creating an External Subroutine
- Session 5 - Editing a Map
- Session 6 - Invoking a Subprogram

Note:

To perform all steps of this tutorial, the database must have been started.

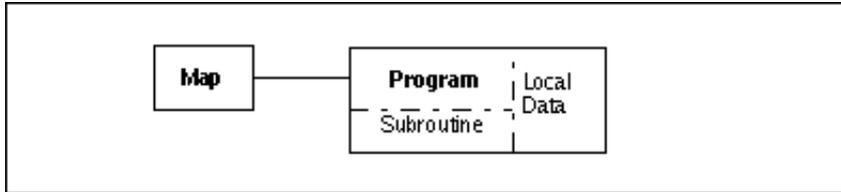
Selecting a Function or Item

There are two ways to select a function or item from a Natural menu or selection list:

1. You select the function/item name with the arrow keys so that the function/item name is highlighted, and then press the ENTER key.
2. You press the alphabetical key that corresponds to the first character of the function/item name. In this case, you need not press ENTER (key-sensitive selection).
For example, to select the Create function from the Map Editor Menu, you simply press the key for the letter "C".

So, when this tutorial uses the wording "select the function xyz", it is up to you to choose the selection method you prefer. To start with, it may be more convenient to use the first selection method so as to get acquainted with the structure and sequence of menus and selection windows. Once you are familiar with that, it is usually faster to use the second method and press a single alphabetical key (instead of scrolling through a whole list to the desired function and then pressing ENTER).

Session 1 - Creating a Program and a Map



In this session, you will use the *program editor* to create a Natural *program*. In this first session, the fields used in the program are defined as local data within the program. Moreover, an inline subroutine is contained within the program.

Also, you will use the *map editor* to create a Natural *map* (screen layout). This map will be invoked by the program and displayed on the screen for the user to enter the input data required for the processing of the program.

Step 1

Invoke Natural according to the procedures at your site. The Natural Main Menu will be displayed:

```

2001-10-25          NATURAL          Library: SYSTEM
12:36:48           V 5.1.1 Software AG 2001  Mode  : REPORT
User: SAG                               Work Area : empty
.....
·Library          Direct          Services          OS          Fin          ·
.....
  
```

In Natural, you can perform a function either by selecting it from a sequence of menus and selection windows (as shown in Step 3), or by entering a Natural system command in the Direct Command window (as shown in Step 2). The system commands that are available are described in the section System Commands.

Step 2

Natural offers two modes of programming: *structured mode* and *reporting mode*.

Generally, it is recommended to use structured mode exclusively, because it provides for more clearly structured applications. Therefore all explanations and examples in this section refer to structured mode. Any peculiarities of reporting mode will not be taken into consideration.

The mode currently in effect is indicated on the Natural Main Menu:

```

2001-10-25          NATURAL          Library: SYSTEM
12:39:05           V 5.1.1 Software AG 2001  Mode  : REPORT
User: SAG                               Work Area : empty
.....
·Library          Direct          Services          OS          Fin          ·
.....
  
```

You must be operating in *structured mode* to work through the sessions in this section.

If the current mode is reporting mode, select "Direct" with the cursor and press ENTER to invoke the Direct Command window:

```

2001-10-25          NATURAL          Library: SYSTEM
 12:43:25          V 5.1.1 Software AG 2001      Mode  : REPORT
User: SAG                                Work Area : empty
.....
·Library          Direct          Services          OS          Fin          ·
.....
          ..... Direct Command .....
          ·
          .....
    
```

In this window, enter the following command:

GLOBALS SM=ON

and press ENTER; the mode changes to structured mode. Then press ESC.

Step 3

Natural user-written applications are stored in *libraries*. It may be necessary to move from one library to another in order to perform a maintenance function or work on a different application.

The ID of the current library is shown on Natural Main Menu:

```

2001-10-25          NATURAL          Library: SYSTEM
 12:47:12          V 5.1.1 Software AG 2001      Mode  : STRUCTURED
User: SAG                                Work Area : empty
.....
·Library          Direct          Services          OS          Fin          ·
.....
    
```

To move to another library, select "Library" on the Main Menu and press ENTER. A selection window will be displayed, listing all libraries:

```

2001-10-25          NATURAL          Library: SYSTEM
 12:50:56          V 5.1.1 Software AG 2001      Mode  : STRUCTURED
User: SAG                                Work Area : empty
.....
·Library          Direct          Services          OS          Fin          ·
.....
.....
· <LOGON>          ·
· DEMO             ·
· DEMO2            ·
· ORDERS           ·
· PRE              ·
· SYSEXP           ·
.....
Select Library
    
```

With the arrow keys, scroll through the list until the library "SYSEXP" is selected, and press ENTER.

The "Library" field in top right-hand corner now shows the new library ID. Moreover a list of all objects in that library is displayed.

Instead of scrolling through the list of libraries to select the desired library, you have the following alternatives:

- select the item <LOGON> at the top of the library list and press ENTER; a window will then be displayed, in which you enter the ID of the desired library and press ENTER.
- select "Direct" from the Main Menu, and then enter the command "LOGON *library-ID*" (*library-ID* being the ID of the desired library) in the Direct Command window and press ENTER.
- Enter the first letter of a library (fast selection) to scroll through a list of libraries beginning with this letter.

Step 4

To create or modify a Natural program, you use the *program editor*.

By default, this is the *Natural* program editor. However, it is also possible to use another editor (this depends on the setting of the profile parameter EDITOR in your Natural parameter file). This tutorial assumes that the Natural program editor is used.

The first item on the list of objects is <DIRECT COMMAND>, and it is already selected with the cursor. Press ENTER to invoke the Direct Command window. Then invoke the program editor by entering the following command in the Direct Command window:

EDIT PGM01 or in short form: E PGM01

- *If you have access to the program "PGM01", the program editor will be invoked and the program "PGM01" be read into the editing area. Make sure that the program matches the one shown below.*
- *If you do not have access to program "PGM01", you will receive an error message upon entering the above command. In this case, enter the following command in the Direct Command window:*

EDIT PROGRAM or in short form: E P

This will invoke the program editor with an empty editing area. Type in the program as shown below.

As you fill up the screen, more blank lines will appear automatically, or you can enter "I" in the prefix area of the editor (where there are usually the line numbers) for more blank lines.

When you have typed in the program, enter the command "CHECK" in the command line of the editor to make sure the program contains no errors. Correct any errors that may be indicated. If necessary, enter the CHECK command again until no further errors are indicated.

Then enter the command "SAVE PGM01" to store the source code of the program under the name "PGM01".

Program PGM01:

```

* Example Program PGM01
* -----
DEFINE DATA
LOCAL
01 #NAME-START      (A20)
01 #NAME-END        (A20)
01 #MARK            (A1)
01 PERSON-VIEW VIEW OF EMPLOYEES
    02 PERSONNEL-ID (A8)
    02 NAME          (A20)
    02 DEPT          (A6)
    02 LEAVE-DUE     (N2)
END-DEFINE
*
REPEAT
*
    INPUT USING MAP 'MAP01'
*
    IF #NAME-START = '.'
        ESCAPE BOTTOM
    END-IF
    MOVE #NAME-START TO #NAME-END
*
    RD1. READ PERSON-VIEW BY NAME
        STARTING FROM #NAME-START
        ENDING AT #NAME-END
        IF LEAVE-DUE >= 30
            PERFORM MARK-SPECIAL-EMPLOYEES
        ELSE
            RESET #MARK
        END-IF
        DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=30' #MARK
    END-READ
*
    IF *COUNTER (RD1.) = 0
        REINPUT 'PLEASE TRY ANOTHER NAME'
    END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
    MOVE '*' TO #MARK
END-SUBROUTINE
END

```

Once you reached this point (either by writing the program yourself or by reading it into the editor), note the following aspects of a Natural program:

- The first statement must always be a DEFINE DATA statement. All variables to be used in the program must be defined in this initial DEFINE DATA statement.
- All statements which initiate a logical construct or processing loop (DEFINE DATA, REPEAT; IF, READ) must be ended with a corresponding END-... statement (END-DEFINE, END-REPEAT, END-IF, END-READ).
- The READ statement is marked with a so-called *statement label*, namely "RD1.". Using this label, it is possible to reference the statement at a later point in the program (see last IF statement).

When this program is executed, a screen (map) is displayed, prompting the user to enter a name. The EMPLOYEES file is searched to locate all employees with that name. Then a report is displayed which includes the name, department and leave due of each employee with that name. Those employees who have 30 or more days leave due are marked with an asterisk.

The prompting screen is invoked using the INPUT USING MAP statement. The output report is formatted according to information in the DISPLAY statement. The processing required to show which employees have 30 or more days leave is handled in the portion of the program starting with "IF LEAVE-DUE...". Those with 30 or more days of leave due have an asterisk in the final report as a result of processing in the PERFORM statement and the DEFINE SUBROUTINE statement.

Step 5

The program contains an INPUT USING MAP statement which invokes a map named "MAP01". This map is yet to be created.

In the command line of the program editor, enter the following command:

EDIT MAP or in short form: E M

and press ENTER. This will invoke the Natural map editor, and the Map Editor Menu will be displayed:

```

.....
.                NATURAL MAP EDITOR (Esc to select field)                .
·Create   Modify   Erase   Drag   Info OFF   Lines   Ops. Map   Quit   ·
.....

Create a new field definition

```

This menu is the main menu of the map editor.

Step 6

The first step is to define a *text constant* on the map.

On the Map Editor Menu, select the "Create" function with the arrow keys, and press ENTER.

The Create selection window will be displayed:

```

A Parameter Data Area
G Global Data Area
H Help Routine
L Local Data Area
M Map
N Subprogram
P Program
S Subroutine
T Text Constant
U User Defined
V View Defined
1 Parm Defined
2 Local Defined
    
```

From this window, you select the type of field you wish to create. For example, it is possible to use fields defined in data areas, help routines, programs, subprograms, subroutines, views (DDMs) and other maps, and include them as fields in the map you are editing.

With the arrow keys, select "Text Constant" from the selection window and press ENTER.

With the cursor position, you determine where you wish a field to be placed on the map. By default, the cursor is placed in the top left-hand corner of the map editing screen.

At this position, enter the following text:

```

PERSONNEL INFORMATION
    
```

When you have finished typing the text, press ENTER.

Press PF2 to display an Attribute/Colour Definition window.

Use the DOWN ARROW or DOWN ARROW keys to scroll through the available display attributes and select "Default" (this will cause the field to be displayed "normally", that is, neither intensified nor in any way highlighted); then press ENTER twice to redisplay the Map Editor Menu.

Step 7

The next step is to add two *Natural system variables* to the map.

On the Map Editor Menu, select the Create function with the arrow keys and press ENTER.

From the Create selection window, select "User Defined" with the arrow keys and press ENTER.

The Extended Field Editing window will be displayed:

```

Extended Field Editing
Field :
Format: A Len:           AL:           PM:           ZP: N  SG: N
Rules : 0 Rule Editing: N Array:       Array Editing: N Mode:
AD:           CD:           CV:           DY: N  HE: N
EM:
    
```

At the same time, a message will prompt you to:

```

Position the cursor and press Enter or format char.(Esc=Cancel)
    
```

It is not possible for two fields to occupy the same position on a map. Therefore position the cursor to an empty location on the map - in this case to the beginning of the line below the text you entered in the previous step - and press ENTER.

A line is displayed indicating the maximum length available for a field, and at the same time a selection window is displayed listing the possible field formats:

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

With the arrow keys, select "System" from the window, and press ENTER. A list of all Natural system variables will be displayed.

Example:

*APPLIC-ID	A8
*APPLIC-NAME	A32
*COM	A128
*CONVID	I4
*CPU-TIME	I4
*CURSOR	N6
*CURS-COL	P3
*CURS-LINE	P3
*DAT4D	A10
*DAT4E	A10
*DAT4I	A10
*DAT4J	A7
*DAT4U	A10
*DATA	N3
*DATD	A8
*DATE	A8
*DATG	A15
*DATI	A8
*DATJ	A5
*DATN	N8
*DATU	A8
*DATV	A11
*DATVS	A9
*DATX	D
*DEVICE	A8
*ERROR-LINE	N4
*ERROR-NR	N7
*ERROR-TA	A8
*ETID	A8
*HARDCOPY	A8
*HARDWARE	A16
*INIT-ID	A8
*INIT-PROGRAM	A8
*INIT-USER	A8
*LANGUAGE	N1
*LEVEL	N2
*LIBRARY-ID	A8

With the arrow keys, scroll through the list to select the system variable DATX, and press ENTER. When the map is executed, this system variable will display the current date.

The system variable will be included as a field in the map, which now looks as follows:

```
PERSONNEL INFORMATION
DD/DD/DD
```

Press ENTER twice and the definition of the system variable field is complete.

Then press ENTER again to redisplay the Map Editor Menu.

Step 8

Repeat the previous step, only this time for the system variable *TIMX. When the map is executed, this system variable will display the current time.

The map should then look as follows:

```
PERSONNEL INFORMATION
DD/DD/DD
TT:TT:TT
```

Press ENTER to redisplay the Map Editor Menu.

Step 9

Select the Create function. From the Create selection window, select "Text Constant".

In the line below the *TIMX field (TT:TT:TT), enter the following text:

```
PLEASE ENTER STARTING NAME :
```

Select a display attribute for the entered text in the same manner as you did for the text "PERSONNEL INFORMATION".

The map now looks as follows:

```
PERSONNEL INFORMATION
DD/DD/DD
TT:TT:TT
PLEASE ENTER STARTING NAME :
```

Press ENTER twice to redisplay the Map Editor Menu.

Step 10

Select the Create function. From the Create selection window, select "User Defined".

The Extended Field Editing window will be displayed:

```
Extended Field Editing
Field :
Format: A Len:           AL:           PM:           ZP: N   SG: N
Rules : 0 Rule Editing: N Array:       Array Editing: N Mode:
AD:           CD:           CV:           DY: N   HE: N
EM:
```

Position the cursor leaving one blank between the text ("PLEASE ENTER STARTING NAME:") and the cursor position and press ENTER.

From the selection window of field formats, select the format "Alphanumeric".

In the Extended Field Editing window, the selected format (A) is now entered and a default name (#1) is assigned to the field.

A message will prompt you to use the cursor keys to size the field to the desired length.

Originally, the length is set to "1" as indicated after the format. You can define the length of the field in the following way:

- Enter the length as a number after the format in the Extended Field Editing window (here, after "AL", Alphanumeric Length).

In this case, enter the length as "20" in the Extended Field Editing window. Press ENTER.

The number of Xs on the map indicates the length of the field.

Now specify a name for the field. Type over the "#1" and enter "#NAME-START" as the field name.

Use the TAB key to select the item "AD=" from the Extended Field Editing window. The Attribute Definition window will be displayed:

```

..Attribute Definition..
·Representation      ·
·Alignment           ·
·I/O Characteristics ·
·Mandatory Characters ·
·Length Characteristics ·
·Upper/Lower Case    ·
·Filler Character    ·
.....
    
```

Select "I/O Characteristics". The following window will be displayed:

```

A  Input, non-protected
M  Output, modifiable
O  Output, protected
    
```

Select "M - Output, modifiable". Press ENTER.

The "M" is added to the attributes in the AD field in the Extended Field Editing window. Press ESC when the setting is correct.

Then select "Filler Character" from the Attribute Definition window.

You are then prompted to enter a filler character. Enter an underscore "_" and press ESC. This will cause any empty positions in the field to be filled with underscores when the map is executed. This enables the user to see the exact position and length of the field, which makes entering input easier.

Press ESC again to finish the attribute definition. The map now looks as follows:

```

PERSONNEL INFORMATION
DD/DD/DD
TT:TT:TT
PLEASE ENTER STARTING NAME: XXXXXXXXXXXXXXXXXXXXXXXX
    
```

Press ESC to redisplay the Map Editor Menu.

Step 11

On the Map Editor Menu, select the option "Ops. Map". A selection window with the following functions will be displayed:

```

C Check Map
E Edit Map
K Key Rules
L List Map
P Prof. Map
R Read Map
S Save Map
T Test Map
W Stow Map
    
```

Select the function "Test Map". Press ENTER.

The map will be tested, and displayed on the screen in the same way as if it were invoked from a program:

```

PERSONNEL INFORMATION
2001-10-25
13:01:22
PLEASE ENTER STARTING NAME: _____
    
```

With the TAB key, move the cursor to the field after "PLEASE ENTER STARTING NAME:", type in a name and press ENTER.

Keep pressing ENTER until testing is finished and the map editing screen is displayed again.

Then press ENTER to redisplay the Map Editor Menu.

Step 12

Now that the map is complete, it must be stored in compiled form, so that it can be invoked by a program.

Select "Ops. Map". From the selection window, select the function "Stow Map". The Stow window will be displayed:

```

...Stow Map As: ....
.Name...: .
.Library: SYSEXP .
.....
    
```

In this window, type in the name "MAP01" and press ENTER to stow the map under that name.

The Stow function stores the map in source and object form; that is, it stores the source code of the map, compiles it, and stores the resulting object module.

Upon completion of the Stow function, the map is displayed in the editor work area.

Press ESC to redisplay the Map Editor Menu.

Then select "Quit" to redisplay the Natural Main Menu.

Step 13

On the Main Menu, select "Direct", and then enter the following command in the Direct Command window:

EDIT PGM01 or in short form: E PGM01

This command invokes the program editor and reads the program PGM01 into the work area.

In the command line at the top of the program editor, enter the command:

RUN or in short form: R

This command compiles and executes the program currently in the work area, PGM01.

A screen will be displayed requesting you to enter a name.

For demonstration purposes, enter the name "MCKENNA". As a result, the message "Please try another name" will be displayed, because the EMPLOYEES file does not contain a record for a person of that name.

Then enter the name "SMITH". Based on this name, the program will produce the following output report:

PAGE	1	2001-10-25 13:10:47		
NAME	DEPARTMENT CODE	LEAVE DUE	>=30	
-----	-----	-----	-----	
SMITH	SYSA05	30	*	
SMITH	MGMT10	12		
SMITH	SALE40	8		
SMITH	MGMT10	8		
SMITH	SALE20	7		
SMITH	MGMT30	8		
SMITH	TECH10	4		
SMITH	TECH10	4		
SMITH	TECH05	8		
SMITH	TECH10	8		
SMITH	TECH10	8		
SMITH	TECH10	4		
SMITH	FINA01	30	*	
SMITH	MGMT01	30	*	
SMITH	FINA01	28		
SMITH	SALE02	28		
SMITH		0		

PAGE	1	2001-10-25 13:14:16		
NAME	DEPARTMENT CODE	LEAVE DUE	>=30	
-----	-----	-----	-----	
SMITH	SYSA05	30	*	
SMITH	MGMT10	13		
SMITH	SALE40	31	*	
SMITH	MGMT10	14		
SMITH	MGMT10	18		
SMITH	SALE20	36	*	
SMITH	MGMT30	22		
SMITH	TECH10	38	*	
SMITH	TECH10	10		
SMITH	SALE20	14		
SMITH	TECH05	20		
SMITH	MGMT10	29		
SMITH	TECH10	33	*	
SMITH	TECH10	12		
SMITH	TECH10	19		
SMITH	FINA01	30	*	
SMITH	MGMT01	30	*	

Keep pressing ENTER until you get to the first screen again. When the program asks you again to enter a name, type a period (.) and delete the remaining characters from the input field. When you press ENTER, you will redisplay the program editor.

Step 14

Whenever you CHECK, RUN, or STOW a program, the program is checked for syntax errors that would keep the program from running.

To introduce such an error, move the cursor to the following statement line:

```
DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=30' #MARK
```

Delete the apostrophe after "30". The line now looks as follows:

```
DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=30 #MARK
```

In the command line, enter the command "CHECK". The following error message will be displayed:

```
NAT0305 Text string must begin and end on the same line.
```

Natural requires that a text string (in this case '>=30') must be begun and closed on the same statement line; the beginning and closing of a text string must be indicated by apostrophes. When the closing apostrophe is deleted, this condition is no longer met.

Step 15

Type in the missing apostrophe again.

Then enter the command "CHECK" again to make sure the program is now correct.

Then enter the command "RUN". When the program has run successfully, display the program again by entering a period ".".

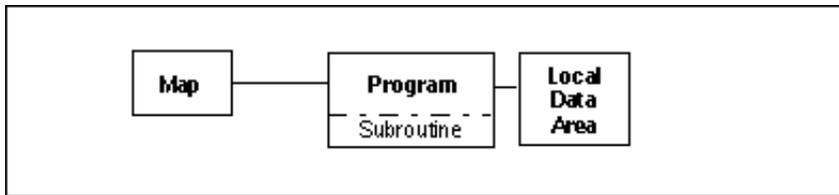
Step 16

Enter the command "STOW" to store the program in source and (compiled) object form.

Then enter a "." in the command line to redisplay the Natural Main Menu.

End of Session 1.

Session 2 - Creating a Local Data Area



In Session 1, the fields used by the program were defined within the DEFINE DATA statement in the program itself. However, it is also possible to place the field definitions in a *local data area* outside the program, with the program's DEFINE DATA statement referencing that local data area by name. For a clear application structure, it is usually better to define fields in data areas outside the programs.

In this session, you will relocate the information in the DEFINE DATA statement to a local data area outside the program. In subsequent sessions some of this information can be used as the basis of a global data area shared by a program and an external subroutine. As you will see in Sessions 3 and 4, an important advantage of data areas is to allow a program and its external subroutine to share the same data in a single data area.

Step 1

Select "Direct" on the Natural Main Menu, and then enter the following command in the Direct Command window:

EDIT LOCAL or in short form: E L

In this editor, you define the data areas to be used by Natural programs or routines.

The *data area editor* will be invoked with the object type set to "LOCAL":

```

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

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

Step 2

Enter the command "I" (for Insert) in the first column of the editor:

```

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

```

The following selection window will be displayed:

```

D Data Field
B Block
C Constant
H Handle
S Structure
U Globally Unique ID
* Comment

```

Select "Data Field". The following window will be displayed:

```

..... Data Field .....
·Level: .
·Field Name: .
·Field Format: .
·Field Length: .
·Arraydefinition: .
·Edit Mask: .
·
·Header Definition: .
·
·Initialization: .
·Value Clause: .
·Comment: .
.....

```

Enter the following to define a variable, using the TAB key to position the cursor to the next field in the window:

```

..... Data Field .....
·Level:      1 .
·Field Name:  #NAME-START .
·Field Format: A .
·Field Length: 20 .
·Arraydefinition: .
·Edit Mask: .
·
·Header Definition: .
·
·Initialization: .
·Value Clause: .
·Comment: .
.....

```

When you have entered all the above, press ENTER.

Then enter the following to define a second variable:

```

..... Data Field .....
·Level:          1
·Field Name:     #NAME-END
·Field Format:    A
·Field Length:   20
·Arraydefinition:
·Edit Mask:
·
·Header Definition:
·
·Initialization:
·Value Clause:
·Comment:
.....
    
```

When you have entered all the above, press ENTER.

Then define a third variable:

```

..... Data Field .....
·Level:          1
·Field Name:     #MARK
·Field Format:    A
·Field Length:   1
·Arraydefinition:
·Edit Mask:
·
·Header Definition:
·
·Initialization:
·Value Clause:
·Comment:
.....
    
```

Press ENTER.

Then press ESC. The local data area now looks as follows:

```

                                Press <ESC> to enter command mode
Mem:          Lib: SYSEXPB   Type: LOCAL   Bytes:   291  Line:    0 of:   3
C T  Comment
*   *** Top of Data Area ***
  1 #NAME-START                A    20
  1 #NAME-END                  A    20
  1 #MARK                       A     1
*   *** End of Data Area ***
    
```

Press ESC and enter "CHECK" in the command line to determine whether the data area contains errors.

Step 3

Now, three variables that are defined in a DDM are to be included in the local data area. Press ESC to enter edit mode and then enter the command "V" as shown below. The following window will be displayed:

```

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

..... View Definition .....
•Name of View:                       .
•Name of DDM:                        .
•Comment:                             .
.....
    
```

Enter the following:

```

Name of View: PERSON-VIEW
Name of DDM:  EMPLOYEES
Comment:
    
```

Then press ENTER.

A window listing all field definitions of the DDM "EMPLOYEES" will be displayed:

```

..... DDM: EMPLOYEES .....
•   1 AA PERSONNEL-ID                A 8 D
•   G 1 AB FULL-NAME
•   2 AC FIRST-NAME                  A 20 N
•   2 AD MIDDLE-I                    A 1 N
•   2 AE NAME                        A 20 D
•   1 AD MIDDLE-NAME                 A 20 N
.....
    
```

You select the fields that are to be included in the local data area by marking them with an "X" in the left-hand column. Mark the following fields: PERSONNEL-ID, NAME, DEPT, and LEAVE-DUE. With the arrow keys, you can scroll through the DDM until these field names are shown on the list. After you have marked all four fields, press ENTER.

The data area now includes the fields selected from the DDM:

```

                                Press <ESC> to enter command mode
Mem:                               Lib: SYSEXP  Type: LOCAL   Bytes: 776  Line: 4 of: 8
C T L Name of View                    Name of DDM
  1 #NAME-START                      A 20
  1 #NAME-END                        A 20
  1 #MARK                           A 1
V  1 PERSON-VIEW                     EMPLOYEES
  2 PERSONNEL-ID                     A 8
  2 NAME                             A 20
  2 DEPT                             A 6
  2 LEAVE-DUE                        N 2.0
*   *** End of Data Area ***
    
```

The local data area is now complete. Press ESC and enter "CHECK" in the command line to check that it contains no errors.

Step 4

A data area must be stored in compiled form before any program referencing the data area can be compiled and executed.

In the command line enter "SAVE LDA01" and press ENTER. Then store the LDA by entering "STOW" in the command line and pressing ENTER.

Step 5

Now that the variables are defined in the local data area LDA01, the DEFINE DATA statement of the program PGM01 has to be changed from actually containing the definitions of variables to merely referencing the local data area in which the variables are defined.

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

E PGM01

This invokes the program editor and reads the program PGM01 into the work area of the editor.

Type a "d" at the beginning of each line (over the line numbers) that defines a variable within the DEFINE DATA statement, as shown below:

d	01	#NAME-START	(A20)
d	01	#NAME-END	(A20)
d	01	#MARK	(A1)
d	01	EMPLOYEES-VIEW	VIEW OF EMPLOYEES
d	02	PERSONNEL-ID	(N8)
d	02	NAME	(A20)
d	02	DEPT	(A6)
d	02	LEAVE-DUE	(N2)

When you press ENTER, these lines are deleted from the program. The lines can also be deleted by using the F4 key and pressing ENTER.

Then enter the following after the word "LOCAL":

USING LDA01

The program should now look as shown below.

Program PGM01:

```

* Example Program PGM01
* -----
DEFINE DATA
  LOCAL USING LDA01
END-DEFINE
*
REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.'
    ESCAPE BOTTOM
  END-IF
*
  MOVE #NAME-START TO #NAME-END
*
  RD1. READ EMPLOYEES-VIEW
        BY NAME
        STARTING FROM #NAME-START
        ENDING AT #NAME-END
*
  IF LEAVE-DUE >= 30
    PERFORM MARK-SPECIAL-EMPLOYEES
  ELSE
    RESET #MARK
  END-IF
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=30' #MARK
*
END-READ
*
IF *COUNTER (RD1.) = 0
  REINPUT 'PLEASE TRY ANOTHER NAME'
END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
*
END

```

Step 6

You can enter comments into a program to document the changes you made to the program. Comments help anyone editing or maintaining a source program, and are ignored in processing.

You mark a line as a comment line by entering either an asterisk (*) or two asterisks (**) at the beginning of the line. In the rest of the line you can then enter any comment. If you wish to append a comment to a line containing a statement, you enter the character string blank-slash-asterisk (/); anything to the right of this character string will then be considered a comment and ignored at execution.

Redisplay the beginning of PGM01 by pressing PAGEUP.

Insert a new empty line by entering an "i" in the first line of the program.

In this empty line, add some comment to indicate that this program has been updated, as shown in the example below.

Example:

```
* Example Program PGM01
* Program now uses a local data area
* -----
DEFINE DATA
  LOCAL USING LDA01 /* this comment is for demonstration purposes only
END-DEFINE
...
```

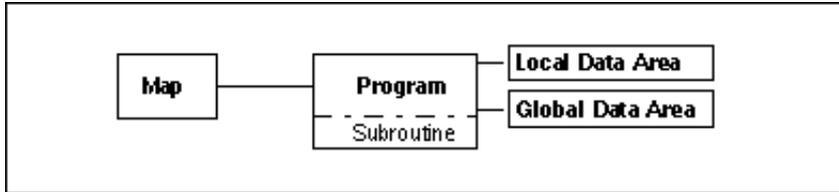
The subsequent sessions in this section show further examples of comments, which you can add to keep track of the changes you have made.

Step 7

CHECK the program and correct any errors. RUN the program to confirm that the results are the same as when the DEFINE DATA statement did not reference a local data area. STOW the program so that it will be available for Session 3.

End of Session 2.

Session 3 - Creating a Global Data Area



In Natural, data can be defined in a single location outside any particular program or routine. Data defined in such a *global data area* can then be shared by multiple programs/routines.

In this session, you will do the following:

- create a global data area;
- modify the local data area created in Session 2;
- modify the program to reference not only the local data area, but also the new global data area.

Step 1

In the command line of the program editor, enter the following command:

E LDA01

The data area editor will be invoked and the local data area LDA01 read into the work area of the editor.

Step 2

To create a new data area, save the contents of the work area under a different name: In the command line of the data area editor, enter the following command:

SAVE GDA01

Then enter the command:

READ GDA01

to read the newly created data area into the work area of the editor. As it is identical to the original one, only the name at the top of the data area editor will change from "LDA01" to "GDA01".

Then enter the command:

SET TYPE GLOBAL

As a result, you will now be editing the data area in the editor as a *global data area*.

Step 3

Press ESC to switch to edit mode and use the line command "d" to delete the first two lines (#NAME-START and #NAME-END). The global data area now looks as follows:

```

                                Press <ESC> to enter command mode
Mem: GDA01      Lib: SYSEXPB      Type: GLOBAL      Bytes: 993 Line: 0 of: 8
C T  Comment
*   *** Top of Data Area ***
  1 #MARK                                A    1
V 1 EMPLOYEE-VIEW                        EMPLOYEES
  2 PERSONNEL-ID                          A    8
  2 NAME                                  A   20
  2 DEPT                                  A    6
  2 LEAVE-DUE                             N   2.0
*   *** End of Data Area ***

```

Press ESC to invoke command mode.

Then enter the command STOW in the command line to store the global data area in source and object form.

Step 4

Now some variables must be removed from the local data area, because they are now defined in the new global data area.

In the command line, enter the command "READ LDA01" to read the local data area LDA01 in the work area of the editor.

Use the "d" line command to delete from LDA01 all variables which are now also defined in GDA01; that is, everything except the two variables #NAME-START and #NAME-END. The modified local data area now looks as follows:

```

                                Press <ESC> to enter command mode
Mem: LDA01      Lib: SYSEXPB      Type: LOCAL      Bytes: 993 Line: 0 of: 8
C T  Comment
*   *** Top of Data Area ***
  1 #NAME-START                          A   20
  1 #NAME-END                            A   20

```

Press ESC and then enter the command STOW.

The modified local data area is now ready to be referenced in the program.

Step 5

As the data to be used by the program PGM01 are now located in two data areas, the DEFINE DATA statement in the program has to be modified so that it references both the global data area GDA01 as well as the local data area LDA01.

In the command line of the data area editor, enter the command "EDIT PGM01" to invoke the program editor and read the program into the work area of the editor.

After the line containing "DEFINE DATA", insert an empty line by entering "i". In the empty line, type in:

```
GLOBAL USING GDA01
```

Then press ENTER twice.

Step 6

Now you will change the output produced by the program: you will add a `WRITE TITLE` statement, and you will modify the `DISPLAY` statement.

The `WRITE TITLE` statement used in this program produces a title on multiple lines in the resulting report. The slash `/"` notation causes a line advance. As nothing else is specified, the title lines will be displayed centered and not underlined.

Insert a `WRITE TITLE` statement above the `DISPLAY` statement, and change the `DISPLAY` statement, as shown below.

Use the `"i"` command to create the empty lines you need for these modifications.

Also, add some comments at the top of the program to indicate the changes you have made.

The revised program - particularly the `DEFINE DATA`, `WRITE TITLE` and `DISPLAY` statements blocks - should now look as shown below.

Program PGM01:

```

* Example Program PGM01
* Program now uses a local data area and a global data area.
* WRITE TITLE has been added, and DISPLAY statement has been changed.
* -----
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.'
    ESCAPE BOTTOM
  END-IF
*
  MOVE #NAME-START TO #NAME-END
*
  RD1. READ EMPLOYEES-VIEW
        BY NAME
        STARTING FROM #NAME-START
        THRU #NAME-END
*
  IF LEAVE-DUE >= 30
    PERFORM MARK-SPECIAL-EMPLOYEES
  ELSE
    RESET #MARK
  END-IF
*
  WRITE TITLE
    / '*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***'
    / '*** ARE MARKED WITH AN ASTERISK ***'//
*
  DISPLAY 23X '//N A M E' NAME
        3X '//DEPT' DEPT
        3X '//LV/DUE' LEAVE-DUE
        3X '//*' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0
    REINPUT 'PLEASE TRY ANOTHER NAME'
  END-IF
*
  END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
END

```

Step 7

Once you have made all changes, CHECK the program and correct any errors if necessary.

Then RUN the program; on the input screen, enter the name "SMITH".

Note the differences in the report output, which now looks as follows:

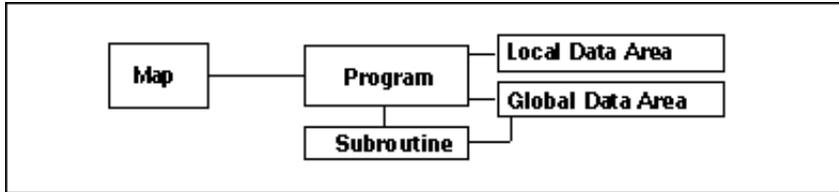
```
*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***
*** ARE MARKED WITH AN ASTERISK ***
```

N A M E	DEPT	LV DUE	*
-----	-----	---	-
SMITH	SYSA05	30	*
SMITH	MGMT10	13	
SMITH	SALE40	31	*
SMITH	MGMT10	14	
SMITH	MGMT10	18	
SMITH	SALE20	36	*
SMITH	MGMT30	22	
SMITH	TECH10	38	*
SMITH	TECH10	10	
SMITH	SALE20	14	
SMITH	TECH05	20	
SMITH	MGMT10	29	
SMITH	TECH10	33	*

When the execution of the program has finished without any errors, STOW the program for the next session.

End of Session 3.

Session 4 - Creating an External Subroutine



In Natural, a *subroutine* can be defined either within a program, or as an external subroutine outside the program.

Until now, the subroutine "MARK-SPECIAL-EMPLOYEES" has been defined within the program using a DEFINE SUBROUTINE statement. In this session, the subroutine will be defined as a separate object external to the program.

Because both internal and external subroutines are invoked with a PERFORM statement, only minimal changes to the program are required.

Step 1

Make a copy of the program PGM01 by saving it under a different name: enter the command "SAVE SUBR01" in the command line of the program editor.

Then enter the command "READ SUBR01" to read the new copy into the work area of the editor.

Enter the command "SET TYPE S" to change the object type from *program* to *subroutine*.

Delete all lines of the subroutine except the comment lines, the DEFINE DATA and DEFINE SUBROUTINE blocks, and the END statement, so that the subroutine looks as follows:

Subroutine SUBR01:

```

* Example Subroutine: SUBR01
* *****
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
END
  
```

CHECK your changes and correct any errors. Then STOW the subroutine.

Step 2

Now that the subroutine is located in SUBR01, the internal subroutine must be removed from program PGM01.

Enter the command "EDIT PGM01".

Delete the following lines, containing the internal subroutine definition, from the program:

```

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

The program now looks as shown below.

Program PGM01:

```

* Example Program PGM01
* Program now uses a local data area and a global data area.
* WRITE TITLE has been added, and DISPLAY statement has been changed.
* The subroutine is now external in SUBR01.
* -----
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.'
    ESCAPE BOTTOM
  END-IF
*
  MOVE #NAME-START TO #NAME-END
*
  RD1. READ EMPLOYEES-VIEW
        BY NAME
        STARTING FROM #NAME-START
        THRU #NAME-END
*
  IF LEAVE-DUE >= 30
    PERFORM MARK-SPECIAL-EMPLOYEES
  ELSE
    RESET #MARK
  END-IF
*
  WRITE TITLE
  / '*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***'
  / '*** ARE MARKED WITH AN ASTERISK ***'//
*
  DISPLAY 23X '//N A M E' NAME
          3X '//DEPT' DEPT
          3X '//LV/DUE' LEAVE-DUE
          3X '//*' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0
    REINPUT 'PLEASE TRY ANOTHER NAME'
  END-IF
*
END-REPEAT
*
END

```

Step 3

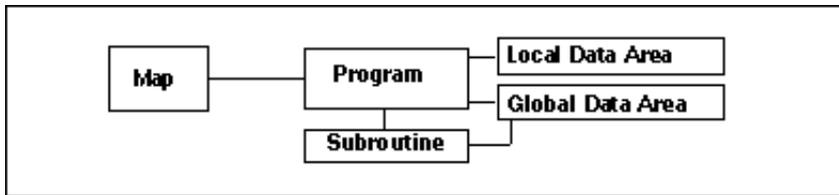
CHECK the program and correct any errors. Then RUN the program to make sure that the results are the same with an external subroutine as previously with an internal subroutine.

STOW the program for the next session.

Then leave the editor and redisplay the Natural Main Menu.

End of Session 4.

Session 5 - Editing a Map



In this session you will edit the screen (map) prompting for an employee name. In short, you will do the following!:

- add a field to enter an ending name for the range of employees;
- modify the layout of the map;
- attach a processing rule to one of the fields in the map;
- assign a help routine to the map.

Step 1

On the Natural Main Menu, select "Direct" and enter the following command in the Direct Command window:

EDIT MAP01

This will invoke the Natural map editor for you to modify the map "MAP01".

On the Map Editor Menu, select "Create".

From the Create selection window, select "Text Constant".

Move the cursor to the line below the one containing "PLEASE ENTER STARTING NAME", and enter the following text:

```
PLEASE ENTER ENDING NAME :
```

Select an attribute for the text constant, and then press ESC to redisplay the Map Editor Menu.

Step 2

Select "Create" and then "User Defined".

Position the cursor leaving one blank after the end of the text constant you just entered and press ENTER.

From the selection window of field formats, select the format "Alphanumeric".

Enter a field length of "20" in the Extended Field Editing window.

Then specify a name for the field: Type over the "#1" and enter "#NAME-END" as the name.

Use the PF2 key to select the item "AD=" from the Extended Field Editing window. The Attribute Definition window will be displayed:

```

..Attribute Definition..
·Representation      ·
·Alignment           ·
·I/O Characteristics ·
·Mandatory Characters ·
·Length Characteristics ·
·Upper/Lower Case    ·
·Filler Character     ·
.....
    
```

Select "I/O Characteristics". The following window will be displayed:

```

A  Input, non-protected
M  Output, modifiable
O  Output, protected
    
```

Select "M - Output, modifiable". Then press ENTER.

Then select "Filler Character" from the Attribute Definition window.

Enter an underscore "_" as filler character and press ESC. Press ENTER to finish the attribute definition.

The map now looks as follows:

```

PERSONNEL INFORMATION
DD/DD/DD
TT:TT:TT
PLEASE ENTER STARTING NAME: XXXXXXXXXXXXXXXXXXXXX
PLEASE ENTER ENDING NAME: XXXXXXXXXXXXXXXXXXXXX
    
```

Step 3

The next step is to change the positions of fields on the map.

Select the text constant "PERSONNEL INFORMATION" with the cursor, and press ESC.

From the Map Editor Menu, select the function "Drag".

With the arrow keys, move the text horizontally to the center of the screen.

Press ENTER to fix the text in its new position.

Then select the field *DATX (DD/DD/DD) move it up one line.

Then move the field *TIMX (TT:TT:TT) up one line.

The map should now look as follows:

```

DD/DD/DD                PERSONNEL INFORMATION
TT:TT:TT

PLEASE ENTER STARTING NAME: XXXXXXXXXXXXXXXXXXXXX
PLEASE ENTER ENDING NAME: XXXXXXXXXXXXXXXXXXXXX
    
```

Press ESC to redisplay the Map Editor Menu.

Step 4

The next step is to create a processing rule and assign it to a map field.

Note:

If you use the OpenVMS TPU editor or the UNIX vi editor as your program editor, you cannot edit processing rules.

From the Map Editor, select the field #NAME-START (that is, the field after the text "PLEASE ENTER STARTING NAME"), enter modify mode, and display the Extended Field Editing window.

With the TAB key, move to "Rule Editing" and enter a "Y" (key-sensitive). The following screen will be displayed:

```

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

```

Two options are available: Rules and Fields.

If Fields were selected, the fields in this map would be listed.

Select "Rules". A selection window will be displayed:

```

..Current Field: #NAME-START.....
.
.           R U L E   E D I T I N G (Esc = Quit)           .
·Rules                                           Fields ·
.....
·<CREATE>·
.....

```

If there were processing rules already assigned to the field, their ranks would be listed in this selection window. As no rules have yet been created for the field, the selection window only contains the item <CREATE>, which allows you to create a new processing rule.

Step 5

Select <CREATE>. The rule editor will be invoked.

You enter the source code for a processing rule in the same way as you enter source code for a program.

At the beginning of the "top of data" line, enter the command "I" to insert an empty line.

Then enter the following processing rule:

```

IF & = ' ' REINPUT 'Please type in a name'
                MARK *&
END-IF

```

Note:

The ampersand (&) in the processing rule will be dynamically substituted by the name of the field to which the processing rule is attached.

Step 6

Up to 100 processing rules (Rank 0 - 99) can be attached to a field. At map execution, the processing rules are executed in ascending order by rank and then by screen position of the field.

In the command line, enter the command "P=1" to assign Rank 1 to the processing rule; at the same time, the rule is saved. The rank will be displayed in the Rules selection window:

```

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

Keep pressing ESC until the Map Editor Menu is displayed again.

Step 7

On the Map Editor Menu, select "Ops. Map", and from the resulting selection window, select the function "Test Map".

The map (including its processing rules) will be tested, and displayed on the screen in the same way as if it were invoked from the program "PGM01".

When you press ENTER without entering anything on the screen, the processing rule will be executed:

```

Please type in a name
2001-10-25           PERSONNEL INFORMATION
13:40:34

PLEASE ENTER STARTING NAME: _____
PLEASE ENTER ENDING NAME:  _____
    
```

Note:

The text "Please type in a name" will be output in the message line. This line may be at the top or bottom of your screen, depending on your NATPARM setting.

In the field after "PLEASE ENTER STARTING NAME", type in a name and press ENTER.

Keep pressing ENTER until testing is finished and the map editing screen is displayed again.

Then press ENTER to redisplay the Map Editor Menu.

Step 8

Select "Create", and from the Create selection window, select "Text Constant".

Below the line containing "PLEASE ENTER ENDING NAME", enter the following text:

```

To stop, type in:
    
```

You are then prompted to select a display attribute for the field. Select "Default".

Then redisplay the Map Editor Menu.

Step 9

Select "Create", and then "Text Constant".

Position the cursor one blank past the text you just entered. Enter a period "." as text.

You are prompted to select a display attribute for the text. The period is to be displayed intensified when the map is executed. Therefore select the attribute "Intensified".

Then redisplay the Map Editor Menu.

Step 10

Repeat the above two steps to create in the next line of the map the following text constant:

```
For help, type in:
```

and after that an intensified question mark "?".

The map now looks as follows:

```
DD/DD/DD                PERSONNEL INFORMATION
TT:TT:TT

PLEASE ENTER STARTING NAME XXXXXXXXXXXXXXXXXXXXXXXX
PLEASE ENTER ENDING NAME XXXXXXXXXXXXXXXXXXXXXXXX
To stop, type in: .
For help, type in: ?
```

Step 11

The next step is to change the order of entire lines of a map.

With the cursor, select the line that contains "To stop ...", and redisplay the Map Editor Menu.

Select the Lines function. The following selection window is displayed:

```
Insert After
Erase Line
Copy After
Duplicate Line
Move After
Split Line
Join Line
```

Select "Move After".

Then with the cursor you select the line after which the line is to be moved. In this case, place the cursor in the line that contains "For help ...", and press ENTER.

The map now looks as follows:

```

DD/DD/DD                PERSONNEL INFORMATION
TT:TT:TT

PLEASE ENTER STARTING NAME XXXXXXXXXXXXXXXXXXXXXXXX
PLEASE ENTER ENDING NAME XXXXXXXXXXXXXXXXXXXXXXXX
For help, type in: ?
To stop, type in: .
    
```

Step 12

The next step is to insert an empty line in the map.

With the cursor, select the line that contains "PLEASE ENTER ENDING NAME", and redisplay the Map Editor Menu.

Select the Lines function, and from the selection window, select "Insert After".

An empty line will be inserted after the selected line:

```

DD/DD/DD                PERSONNEL INFORMATION
TT:TT:TT

PLEASE ENTER STARTING NAME XXXXXXXXXXXXXXXXXXXXXXXX
PLEASE ENTER ENDING NAME XXXXXXXXXXXXXXXXXXXXXXXX

For help, type in: ?
To stop, type in: .
    
```

Step 13

The next step is to assign a helproutine to the field #NAME-START.

With the cursor, select the field #NAME-START (the field after "PLEASE ENTER STARTING NAME"), and press ENTER to redisplay the Map Editor Menu.

Select "Modify". The Extended Field Editing window for the selected field will be displayed.

With the TAB key, select "HE=". A window will be displayed for you to enter the name of a helproutine. Enter the following name (in apostrophes!):

```
'HELP01'
```

"HELP01" - which is yet to be created - is now assigned as helproutine to the field.

Step 14

Keep pressing ENTER until the Map Editor Menu reappears.

Select "Ops. Map", and from the selection window, select the function "Stow Map". The Stow window will be displayed:

```

...Stow Map As: ....
.Name: MAP01      .
.Libr: SYSEXP    .
.....
    
```

As the map name "MAP01" is already entered, just press ENTER to stow the map.

Upon completion of the Stow function, the map is displayed in the work area of the editor.

Redisplay the Map Editor Menu.

Step 15

Now the helproutine "HELP01" has to be created.

Select "Quit" to redisplay the Natural Main Menu.

Select "Direct", and in the Direct Command window, enter the following command to create a new map:

EDIT MAP or in short form: E M

The Map Editor Menu will be displayed. Select "Create", and then "Text Constant".

Enter the first line of the following text:

Type in the name of an employee in the first field and press ENTER. You will then receive a list of all employees of that name. If you enter names in both fields, you will get a list of all employees in alphabetical order from the first specified name to the second specified name.

Then repeat the creation of text constants, one for each line of text, until all of the above text has been entered.

Step 16

Then redisplay to the Map Editor Menu.

Select "Ops. Map", and then "Prof. Map". The Map Settings window will be displayed:

```

.Map Settings.....
.Format                Context
-----
.Page Size .....: 23
.Line Size .....: 79      WRITE Statement ...: N
.Layout .....:
. dynamic .....: N      Help Routine .....:
.Zero Print ....: N      Help Parameter ....:
.Upper Case ....: Y      as field default .: N
.Documentation Skip ...: N      Help Text .....: N
.Decimal Char ..: .      Position Line ....: 0      Column : 0
.Standard Keys .: N      AutoRuleRank .....: 1
.Right Justify .: Y
.Print Mode ....:      Filler Characters
.-----
.Control Var ...:      Optional, Partial .: _
.                      Required, Partial .: _
.Field Sensitive: N      Optional, Complete .: _
.                      Required, Complete .: _
.....

```

The page size is set to 23, the line size to 79.

Change the page size to 15 and the line size to 25 by typing over the existing values.

Press ENTER, and you are prompted:

```
Warning Please confirm lines columns may be lost (Y/N).
```

Enter "Y" (key-sensitive), and the editor reappears. Press ENTER for the map menu and select the function "Stow Map".

In the Stow window, enter the name "HELP01" to store the new map.

Upon completion of this Stow function, the map is displayed in the work area of the editor.

Redisplay the Map Editor Menu.

Then select "Quit" to redisplay the Main Menu.

Step 17

Now the map MAP01 has to be tested to see if the attached help map HELP01 is executed correctly.

Select "Direct" and enter the following command in the Direct Command window:

E MAP01

The Map Editor Menu for MAP01 will be displayed.

Select "Ops. Map", and then "Test Map".

When you enter a question mark (?) in the field after "PLEASE ENTER STARTING NAME", the help routine or help map assigned to that field should execute and the help text entered in Step 15 displayed.

Press ENTER twice, and the processing rule attached to the first field will be executed, prompting you to "Please type in a name".

Enter a name to end testing. The map editing screen will be displayed.

Redisplay the Map Editor Menu, and from there with "Quit", redisplay the Main Menu.

Step 18

Now the program PGM01 has to be adjusted to the modified map.

Select "Direct" and enter the command "E PGM01" in the Direct Command window to read PGM01 into the work area of the program editor.

Until now, the program included the instruction:

```
MOVE #NAME-START TO #NAME-END
```

Thus the start value for the list displayed by the program was also the end value, which meant that in the example used the list contained only employees whose names were "SMITH". (Otherwise, all employees from "SMITH" to the end of the alphabet would have been included in the report.) Now the map allows you to specify both a start value *and* an end value for the list to be output. However, an IF statement has to be added to the program to handle a situation in which no end value is specified.

Change the program so that the MOVE statement is contained within an IF statement block as follows:

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

The program should now look as shown below.

Program PGM01:

```

* Example Program PGM01
* Program now uses a local data area and a global data area.
* WRITE TITLE has been added, and DISPLAY statement has been changed.
* The subroutine is now external in SUBR01.
* -----
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.'
    ESCAPE BOTTOM
  END-IF
*
  IF #NAME-END = ' '
    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 >= 30
    PERFORM MARK-SPECIAL-EMPLOYEES
  ELSE
    RESET #MARK
  END-IF
*
  WRITE TITLE
  / '*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***'
  / '*** ARE MARKED WITH AN ASTERISK ***'//
*
  DISPLAY 23X '//N A M E' NAME
          3X '//DEPT' DEPT
          3X '/LV/DUE' LEAVE-DUE
          3X '//*' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0
    REINPUT 'PLEASE TRY ANOTHER NAME'
  END-IF
*
END-REPEAT
*
END

```

Step 19

CHECK the program and correct any errors that may be indicated.

Then RUN the program. On the input screen, enter the names "JONES" and "JUNG" as start value and end value respectively.

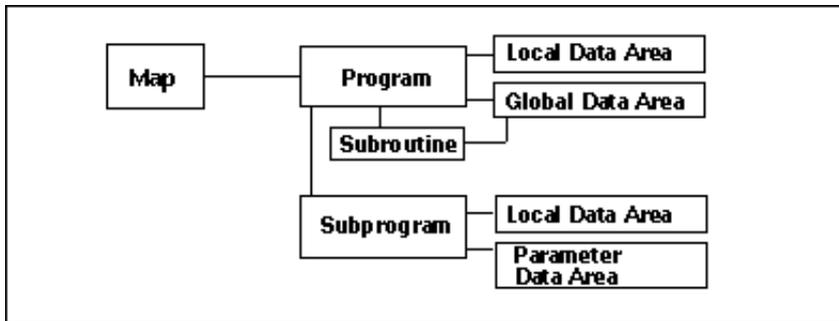
Then type "." in the first field and press ENTER to stop the execution of the program.

After the program has been executed, STOW it for the next session.

Then redisplay the Natural Main Menu.

End of Session 5.

Session 6 - Invoking a Subprogram



In Natural, both *subprograms* and *subroutines* can be invoked from a program. They differ from one another in the way data from the invoking program can be accessed.

As shown in Session 4, a *subroutine* can access the same global data area as the invoking program. However, a *subprogram* is invoked with a CALLNAT statement, and with this statement, parameters can be passed from the invoking program to the subprogram. These parameters are the only data available to the subprogram from the invoking program.

The passed parameters must be defined either within the DEFINE DATA PARAMETER statement of the subprogram, or in a parameter data area used by the subprogram.

In addition, a subprogram can have its own local data area, in which the fields to be used within the subprogram are defined.

In this session, you will create a subprogram with a parameter data area and a local data area. Moreover, a CALLNAT statement to invoke the subprogram will be added to the main program; also the main program's local data area has to be modified. In the subprogram, the employees selected by the main program will be the basis to select the corresponding vehicle information from the VEHICLES file. As a result, the report will contain vehicle information as well as employee information.

Step 1

First, the main program's local data area has to be modified.

On the Main Menu, select "Direct" and enter the command "E LDA01" in the Direct Command window.

The data area editor will be invoked and the local data area LDA01 will be read into the work area:

```

Press <ESC> to enter command mode
Mem: LDA01      Lib: SYSEXPB  Type: LOCAL      Bytes: 639  Line: 0 of: 5
C T  Comment
*   *** Top of Data Area ***
  1 #NAME-START          A  20
  1 #NAME-END            A  20
*   *** End of Data Area ***
    
```

Move the cursor down to highlight "#NAME-END" and type "i" to insert a line. Add the variables #PERS-ID, #MAKE and #MODEL to the local data area, as shown below. They will be used in the CALLNAT statement to be added to the program.

The local data area now looks as follows:

```

                                Press <ESC> to enter command mode
Mem: LDA01      Lib: SYSEXPB   Type: LOCAL      Bytes: 639 Line: 0 of: 5
C T  Comment
*   *** Top of Data Area ***
  1 #NAME-START                A   20
  1 #NAME-END                  A   20
  1 #PERS-ID                   A    8
  1 #MAKE                      A   20
  1 #MODEL                    A   20
*   *** End of Data Area ***
    
```

CHECK the local data area and correct any errors. Then STOW it.

Step 2

A parameter data area is needed for the subprogram. With minor modifications it is possible to use the local data area LDA01 to create this parameter data area.

Make a copy of LDA01 by saving it under a different name: in the command line of the editor, enter the command "SAVE PDA02".

Then enter the command "READ PDA02" to read the new copy into the editor work area.

Then enter the command "SET TYPE PARAMETER" to change the data area type from "LOCAL" to "PARAMETER".

With the line command "d", delete the first two lines. The parameter data area now looks as follows:

```

                                Press <ESC> to enter command mode
Mem: PDA02      Lib: SYSEXPB   Type: PARAMETER Bytes: 445 Line: 1 of: 3
C T L Name of Datafield          F Leng Index/Comment      M
*   *** Top of Data Area ***
  1 #PERS-ID                A    8
  1 #MAKE                   A   20
  1 #MODEL                  A   20
*   *** End of Data Area ***
    
```

CHECK the parameter data area and correct any errors. Then STOW it.

Step 3

Now, a local data area for the subprogram has to be created.

In the command line of the editor, enter the command "CLEAR" to clear the work area.

Then enter the command "SET TYPE LOCAL" to change the data area type.

Switch to edit mode by pressing ESC and then enter a "V" in the first column of the screen.

The View Definition screen will be displayed. Enter the following:

```

..... View Definition .....
·Name of View: CAR-VIEW      ·
·Name of DDM:  VEHICLES    ·
·Comment:      ·
.....
    
```

Press ENTER. A window listing all field definitions of the DDM "VEHICLES" will be displayed.

Select the fields that are to be included in the data area by marking them with an "X" in the left-hand column. Mark the following fields: PERSONNEL-ID, MAKE and MODEL After you have marked all three fields, press ENTER.

The local data area now looks as follows:

```

                                Press <ESC> to enter command mode
Mem:                               Lib: SYSEXP  Type: LOCAL      Bytes: 632  Line: 0 of: 5
C T  Comment
*   *** Top of Data Area ***
V 1  CAR-VIEW                               VEHICLES
   2  PERSONNEL-ID                          A    8
   2  MAKE                                   A   20
   2  MODEL                                  A   20
*   *** End of Data Area ***

```

CHECK the local data area and correct any errors.

Then stow it by entering the command "STOW LDA02".

Redisplay the Natural Main Menu.

Step 4

The next step is to create the subprogram itself.

Invoke the program editor with the EDIT command, write the subprogram shown below, and STOW it under the name "SPGM02".

Subprogram SPGM02:

```

* Example Subprogram 'SPGM02'
* *****
DEFINE DATA
  PARAMETER USING PDA02
  LOCAL      USING LDA02
END-DEFINE
*
FD1. FIND (1) CAR-VIEW WITH PERSONNEL-ID = #PERS-ID
      MOVE MAKE (FD1.)   TO #MAKE
      MOVE MODEL (FD1.) TO #MODEL
      ESCAPE BOTTOM
END-FIND
END

```

This subprogram receives the personnel number passed by the main program and then uses this number as the basis of a search of the VEHICLES file.

Step 5

Now, the main program PGM01 has to be modified to invoke the subprogram.

Read the program into the editor, and insert the following statements immediately before the WRITE TITLE statement:

```

RESET #MAKE #MODEL
CALLNAT 'SPGM02' PERSONNEL-ID #MAKE #MODEL

```

Some of the parameters passed in the CALLNAT statement are defined in the global data area, and some in the local data area.

Please note that the variables defined in the parameter data area of the subprogram need not have the same name as the variables in the CALLNAT statement; it is only necessary that they match in sequence, format, and length.

As the program receives vehicle information from the subprogram, which is also to be output in the report, the DISPLAY statement has to be modified as shown below.

The program should then look as follows:

Program PGM01:

```

* Example Program PGM01
* Program now uses a local data area and a global data area.
* WRITE TITLE has been added, and DISPLAY statement has been changed.
* The subroutine is now external in SUBR01.
* A subprogram provides vehicle information.
* -----
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.'
    ESCAPE BOTTOM
  END-IF
*
  IF #NAME-END = ' '
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ PERSON-VIEW BY NAME
        STARTING FROM #NAME-START
        ENDING AT #NAME-END
*
    IF LEAVE-DUE >= 30
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
  RESET #MAKE #MODEL
  CALLNAT 'SPGM02' PERSONNEL-ID #MAKE #MODEL
*
  WRITE TITLE
    / '*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***'
    / '*** ARE MARKED WITH AN ASTERISK ***'//
*
  DISPLAY      '//NAME' NAME
              2X '//DEPT'  DEPT
              2X '//LV/DUE' LEAVE-DUE
              ' ' #MARK
              2X '//MAKE'  #MAKE
              2X '//MODEL' #MODEL
*
  END-READ
*
  IF *COUNTER (RD1.) = 0
    REINPUT 'PLEASE TRY ANOTHER NAME'
  END-IF
END-REPEAT
END

```

Step 6

After you have made all changes to the program, CHECK it and correct any errors; then RUN it.

Then STOW it for future reference.

End of Session 6.