

Tutorial - Getting Started with Natural

The sample sessions provided in this section are based on a program, a map, and a subprogram. Ask your Natural administrator to make these available to you. If you do not have access to copies of these objects, you will still be able to step through these sessions by creating them yourself.

The output screens provided in the sessions are merely meant as examples and may not correspond with your results. Also, behavior and appearance of Natural (screen layout, system messages, etc.) may differ from your environment as they depend on the system parameters set by your Natural administration.

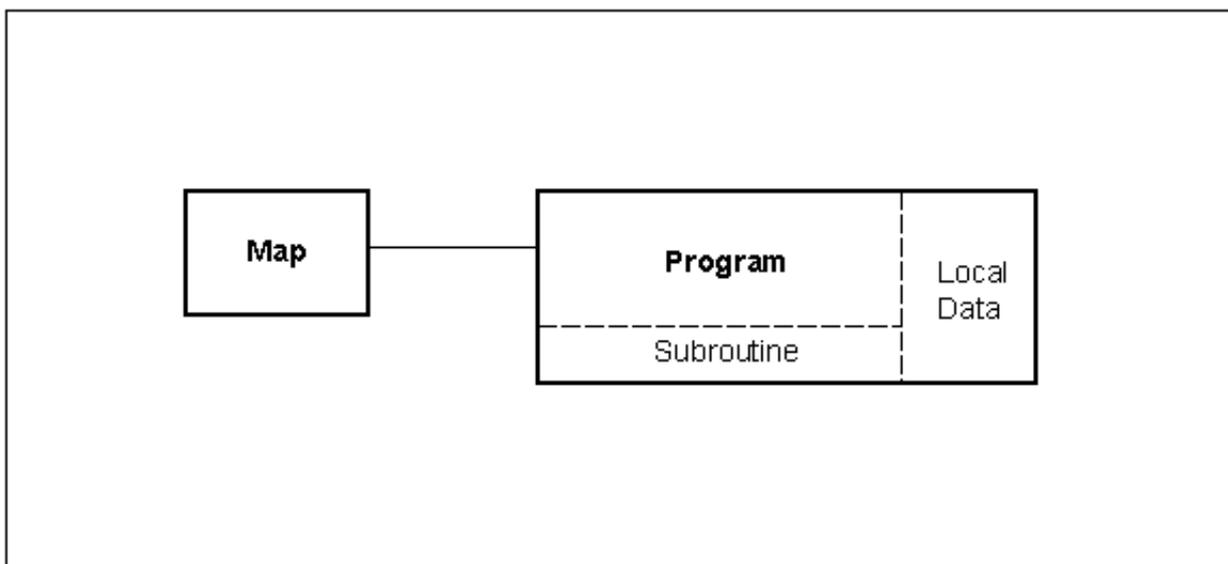
These sessions are **not** intended to provide an example of how an application should be built. Rather, each session is designed to provide a gradual exposure to specific features and build a context for introducing the next session's features. It is important that you work through the sessions in the sequence below. This tutorial approach does not parallel a typical application development cycle.

- 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

See also:

- Tutorial - Using the Map Editor
-

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 *map* used by the program.

Step 1

Invoke Natural according to the procedures at your site. If you get a NEXT or MORE prompt, enter the command MAINMENU. The Natural Main Menu will be displayed:

```

16:50:53                ***** NATURAL *****                2001-01-30
User SAG                 - Main Menu -                          Library SYSTEM

                        Function

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

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

```

Step 2

Both Natural system programs and 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 field "Library" in the top right-hand corner shows the ID of the current library. To move to another library, enter the command "LOGON *library-ID*" (*library-ID* being the ID of the library you want to access) in the Command line. If you have access to a library that contains copies of the sample programs used in these sessions, enter the *library-ID* of that library. By default, sample programs are provided in the system library SYSEXP; ask your Natural administrator for details.

Step 3

On the Natural Main Menu, select "Development Functions". The Development Functions menu will be displayed:

```

16:51:14          ***** NATURAL *****          2001-01-30
User SAG          - Development Functions -          Library SYSTEM
                                                Mode Reporting
                                                Work area empty

Code  Function

C    Create Object
E    Edit Object
R    Rename Object
D    Delete Object
X    Execute Program
L    List Object(s)
S    List Subroutines Used
?    Help
.    Exit

Code .. _      Type .. _
                Name .. _____

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

```

You must be operating in *structured mode* to work through these sessions. If your session is in *reporting mode*, change the mode by entering an "S" in the first position of the "Mode" field. (If you should not be able to change the mode in this manner - it is dependent on your site - contact your Natural administrator for assistance.)

Step 4

In the course of developing application systems, Natural objects can be created and modified from the Development Functions menu.

If you wish to get help information about the functions on this menu, enter a question mark (?) in the Code field.

Below the list of functions, there are three input fields: "Code", "Type", and "Name". Further below is the command line. You can perform a Natural function either by entering the appropriate values in the input fields, or by entering a system command in the command line.

For example, to invoke the program editor to edit an existing program PGM01, you would enter the following in the input fields:

	?	Help
	.	Exit
Code ..	E	Type .. _
		Name .. PGM01 _____
Command ==>		
Enter-PF1---	PF2---	PF3---
PF4---	PF5---	PF6---
PF7---	PF8---	PF9---
PF10--	PF11--	PF12---
Help	Exit	Canc

The equivalent system command, entered in the command line, would be:

	?	Help
	.	Exit
Code ..	_	Type .. _
		Name .. _____
Command ==> EDIT PGM01		
Enter-PF1---	PF2---	PF3---
PF4---	PF5---	PF6---
PF7---	PF8---	PF9---
PF10--	PF11--	PF12---
Help	Exit	Canc

If an object already exists (such as, program PGM01 in the previous example), it is not necessary to specify its type in the "Type" field; this is because each object in a Natural library, regardless of its type (program, map, subroutine, etc.), must have a unique name. When you create a new object, you have to specify the type of the object in the "Type" field.

Once you have become familiar with the sequence of menu screens, you will be able to go directly to the screen you want by issuing a system command. You may enter a system command on every Natural screen which provides a command line.

Step 5

On the Development Functions menu, enter the code "E". The Natural *program editor* will be displayed:

```

>
All      > + Program                               Lib SYSTEM
.....1.....2.....3.....4.....5.....6.....7..
0010
0020
0030
0040
0050
0060
0070
0080
0090
0100
0110
0120
0130
0140
0150
0160
0170
0180
0190
0200
.....1.....2.....3.....4.....5..... S 0   L 1

```

- If you have access to a copy of program PGM01, enter the command READ PGM01 in the command line at the top of the editor screen. Make sure that the program matches program PGM01 shown on the following page.
- If you do not have access to a copy of PGM01, type in the program as illustrated in the following section. (If the source work area is not empty, enter the command CLEAR in the command line at the top of the editor screen.) As you fill up a screen, type the command ADD in the command line for more blank lines. When complete, enter the command SAVE PGM01 in the command line to store the program.

Note:

To return to the beginning of the program, enter the command TOP in the editor's command line, to go to the end of the program, enter the command BOT.

Program PGM01:

```

* Example Program 'PGM01' for User's Guide Tutorial
* -----
DEFINE DATA
LOCAL
01 #NAME-START      (A20)
01 #NAME-END        (A20)
01 #MARK            (A1)
01 EMPLOYEES-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 EMPLOYEES-VIEW BY NAME
          STARTING FROM #NAME-START
          THRU #NAME-END
IF LEAVE-DUE >= 20
    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
  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 this example 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. In this example program, the variables are defined in a DEFINE DATA LOCAL statement; that is, the data are defined *internal* to the program.
- 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.". Via 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 appears, 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 more than 20 days leave due are marked with an asterisk (*).

The prompting screen is invoked via the INPUT USING MAP statement. The report is formatted according to information in the DISPLAY statement. The processing required to show which employees have more than 20 days leave is handled in the portion of the program starting with "IF LEAVE-DUE ...". Those with 20 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 6

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

In the program editor's command line, enter a period (.) to return to the Development Functions menu. On the Development Functions menu, enter the function code "E" (for Edit Object) in the Code field and "M" (for Map) in the Type field:

```

      .      Exit
      Code .. E      Type .. M
      Name .. _____

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

The Edit Map menu will be displayed:

```

17:55:11          ***** NATURAL MAP EDITOR *****          2001-01-30
User SAG          - Edit Map -          Library SYSTEM

      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
    
```

Step 7

Enter the code "I" (Initiate a New Map) and the name MAP01. The Define Map Settings For Map screen will be displayed:

Delimiters				Format	Context
17:56:47				Define Map Settings for MAP	2001-01-30
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 _	-----
				Control Var _____	Optional, Partial
					Required, Partial
					Optional, Complete ...
				Apply changes only to new fields? N (Y/N)	Required, Complete ...
Enter-PF1---	PF2---	PF3---	PF4---	PF5---	PF6---
PF7---	PF8---	PF9---	PF10--	PF11--	PF12---
Help		Quit			Let

Step 8

In the Filler Characters section of the screen, enter an underscore (_) after each of the options:

Justification .. L (L/R)	Filler Characters
Print Mode _	-----
	Optional, Partial _
Control Var _____	Required, Partial _
	Optional, Complete ... _
Apply changes only to new fields? N (Y/N)	Required, Complete ... _
Enter-PF1---	PF2---
PF3---	PF4---
PF5---	PF6---
PF7---	PF8---
PF9---	PF10--
PF11--	PF12---
Help	Quit
	Let

The map will use this character whenever a field has empty positions to fill within a field. (Delimiter characters can also be modified on this screen, however for these sessions, they are left unchanged.)

Press ENTER again. The map editor screen will be displayed in split-screen format:

```

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
    
```

The top portion of the screen will not be used in this exercise. The bottom section is the editing area in which you will create the map.

During the creation of the map, you will use a number of map editor *line commands* and *field commands*.

- A line command begins with two periods (..). You enter it at the beginning of a line, and it applies to the whole line in which you enter it.
- A field command begins with one period (.). You enter it at the beginning of a field, and it applies only to the field in which you enter it.

(The sessions in the section Tutorial - Using the Map Editor also show you how to apply these commands to more than one line/field at a time.)

Step 9

Move the cursor to the second line of the editing area and type in the following:

Please enter starting name :X(20)

When you press ENTER, the screen 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---+-----

Please enter starting name :XXXXXXXXXXXXXXXXXXXXX

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

If the letters you typed in are automatically converted to upper case, press PF3 to return to the Edit Map menu and enter "%L" in the command line.

Step 10

Type in the line command ".C" (Center) over the first three positions of the line that contains "Please enter starting name" as shown below:

```

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---+-----

..case enter starting name :XXXXXXXXXXXXXXXXXXXXX
    
```

As a result, the line will be centered:

```

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---+-----
                                     Please enter starting name :XXXXXXXXXXXXXXXXXXXXX
    
```

Step 11

Move the cursor to the end of the new line, leave a space after the field and type in the following text:

(. to exit)

The screen will 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---+-----
                                     Please enter starting name :XXXXXXXXXXXXXXXXXXXXX (. to exit)

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

Step 12

Type in the field command ".E" (for extended field editing) over the first two positions of the field "XXXXXXXXXXXXXXXXXXXX", as shown below:

```

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---+-----

Please enter starting name .XXXXXXXXXXXXXXXXXXXXX (. to exit)
    
```

This will invoke the extended field editing section for the field marked with the command:

```

Fld #001                                           Fmt A20
-----
AD= MIT'_'_____ ZP= OFF      SG= OFF      HE= _____ Rls 0
AL= _____    CD= ___      CV= _____ Mod Undef
PM= ___ DF=       DY= _____
EM= _____
001  --010---+-----+-----030---+-----+-----050---+-----+-----070---+-----

Please enter starting name .XXXXXXXXXXXXXXXXXXXXX (. to exit)
    
```

The field "Fld" in the upper left corner contains the field name "#001". This number has been assigned automatically by Natural.

Overwrite "#001" with the field name "#NAME-START":

```

Fld #NAME-START                                   Fmt A20
-----
AD= MIT'_'_____ ZP= OFF      SG= OFF      HE= _____ Rls 0
AL= _____    CD= ___      CV= _____ Mod Undef
PM= ___ DF=       DY= _____
EM= _____
001  --010---+-----+-----030---+-----+-----050---+-----+-----070---+-----
    
```

Press PF3 to leave the extended field editing section.

Step 13

The map is now complete.

Press PF4 to test the map. The map will be displayed in the form it which it will be displayed on the screen it is invoked via the INPUT USING MAP statement in the program PGM01:

```
Please enter starting name _____ (. to exit)
```

Press PF3 to stop testing and return to the map editing screen.

Step 14

Press PF3 again to return to the Edit Map menu.

Enter the code "W". The map is now stowed (that is, stored in source form and in object form) and ready to be used by the program.

To return to the Development Functions menu, enter a period (.) in the Code field.

Step 15

In the command line of the Development Functions menu, enter the command EDIT PGM01.

Then enter the command RUN in the command line at the top of the program editor. This command compiles and executes the program PGM01.

A screen will be displayed requesting you to enter a name. For demonstration purposes, enter the name JONES.

Based on this name, the program will produce the following output report:

```
Page          1                                00-11-30  12:45:56
      NAME                DEPARTMENT  LEAVE  >=20
                   CODE          DUE
-----
JONES                SALE30         25     *
JONES                MGMT10         34     *
JONES                TECH10         11
JONES                MGMT10         18
JONES                TECH10         21     *
JONES                SALE00         30     *
JONES                SALE20         14
JONES                COMP12         26     *
JONES                TECH02         25     *
```

Step 16

Keep pressing ENTER until you get to the map input screen again. When the program asks you again to enter a name, enter a period (.) and delete the remaining characters from the input field. You will be returned to the program editor.

Step 17

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

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

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

Delete the apostrophe after "20". The line now look as follows:

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

In the command line, enter the command CHECK. The error message "Text string must begin and end on the same line." will appear:

```

>                                     > + Program   PGM01   Lib SYSEXP
...+...1...+...2...+...3...+...4...+...5...+...6...+...7...
0350          BY NAME
0360          STARTING FROM #NAME-START
0370          THRU #NAME-END
0380 *
0390  IF LEAVE-DUE >=20
0400      PERFORM MARK-SPECIAL-EMPLOYEES
0410  ELSE
0420      RESET #MARK
0430  END-IF
0440 *
E 0450  DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20 #MARK
0460 *
0470  END-READ
0480 *
0490  IF *COUNTER (RD1.) = 0
0500      REINPUT 'PLEASE TRY ANOTHER NAME'
0510  END-IF
0520 *
0530  END-REPEAT
0540 *
...+...1...+...2...+...3...+...4...+...5...+... S 59   L 35
NAT0305 Text string must begin and end on the same line.

```

Natural requires that a text string (in this case '>=20') 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.

Note:

If you wish to get more information explaining the meaning of the error message, you can enter a question mark (?) and the error number of the message in the command line to invoke the help system, for example, "? NAT0305".

Step 18

Type in the missing apostrophe again. Then enter the command `CHECK` in the command line again to make sure the program is now correct.

Then enter the command `RUN` in the command line. When the program has run successfully, return to the program editor again by entering a period (`.`).

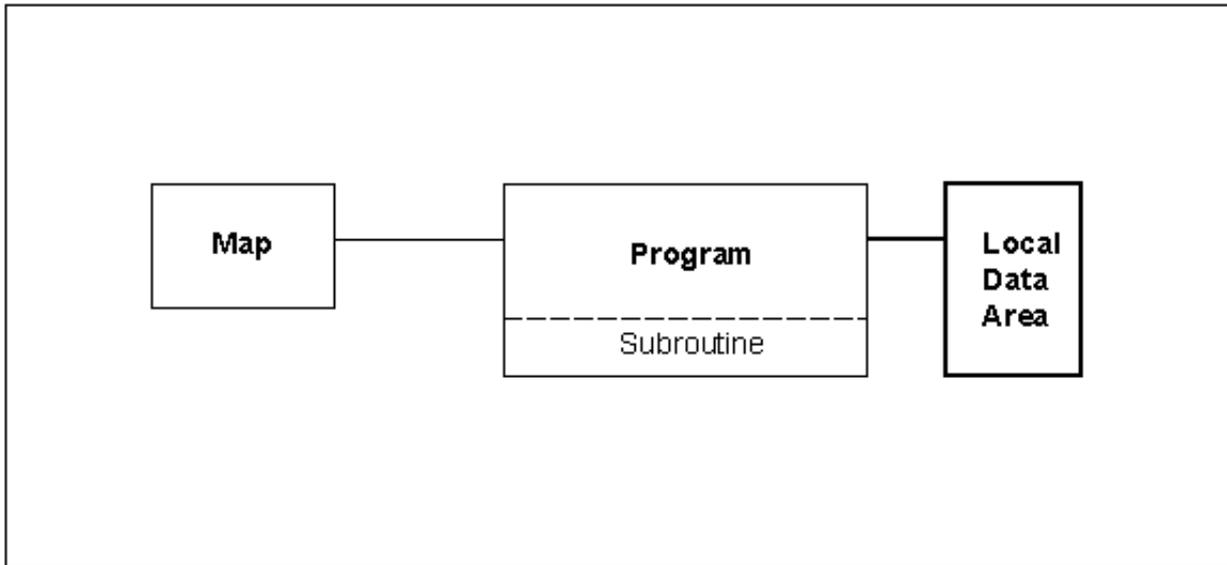
Step 19

Enter the command `STOW` in the command line to compile the program and save both source and object form.

Then enter a period (`.`) in the command line to return to the Development Functions 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. 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.

In this session, the information in the DEFINE DATA statement will be relocated 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.

Several program editor commands will be used in this and the following sessions. See the section Editor Commands for a detailed explanation.

Step 1

On the Development Functions menu, enter the code "E" and object type "L" to create a *local data area*.

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

```

Local                Library SYSTEM                DBID  10 FNR  32
Command                > +
I T L Name                F Leng Index/Init/EM/Name/Comment
All - -----

```

----- S 0 L 1

In this editor, you can define the data areas to be used by Natural programs or routines. A basic outline of the fields on this screen is provided below:

Field	Explanation
I	This is an <i>information</i> field used by the editor to indicate the presence of a definition error ("E"), or to indicate other information on the variable (initial value, edit mask definition, etc.) You cannot modify information in this field.
T	This field indicates the <i>type</i> of the variable. For example, some fields in the data area to be created in this session will be part of a view, which will be indicated by a "V" in this column. Other object types include groups, data blocks, multiple-value fields, periodic groups, and constants.
L	In this field you specify the hierarchical <i>level</i> of the variable (1 to 9). A variable which is not within a hierarchical structure is assigned a level 1 designation.
Name	In this field you specify the <i>name</i> of the variable (or block or view).
F	In this field you specify the <i>format</i> of the variable.
Leng	In this field you specify the <i>length</i> of the variable.
Index/...	This field is used for array definition, initial value assignment, edit mask information, originating view name (for fields derived from a view), or a comment.

Step 2

In the command line of the data area editor, enter the command "SPLIT P PGM01". This puts the editor into split-screen mode with program PGM01 appearing in the lower half of the screen and the data area (as yet blank) in the upper half.

To scroll forward in the lower half of the screen, enter the command "SPLIT +" in the command line of the data area editor ("SPLIT -" scrolls backward). Enter the command again until the DEFINE DATA LOCAL statement and at least three statement lines below it are displayed on the screen:

```

Local          Library SYSTEM                      DBID  10 FNR  32
Command                                             > +
I T L Name          F Leng Index/Init/EM/Name/Comment
All - -----
----- S 0      L 1
Program      PGM01      Library SYSTEM
 0120  DEFINE DATA
 0130    LOCAL
 0140    01 #NAME-START          (A20)
 0150    01 #NAME-END           (A20)
 0160    01 #MARK                (A1)
 0170    01 EMPLOYEES-VIEW VIEW OF EMPLOYEES

```

Step 3

Using the program as a reference, copy the definitions of the first three variables from the DEFINE DATA statement into the data area editor screen. In column "L" enter the level number "1" before each variable; in column "Name" enter the variable names; in column "F" enter their formats, and in column "Leng" their lengths.

The screen should now look as follows:

```

Local          Library SYSTEM                      DBID  10 FNR  32
Command
I T L Name          F Leng Index/Init/EM/Name/Comment  > +
All - -----
    1 #NAME-START    A   20
    1 #NAME-END      A   20
    1 #MARK          A    1

----- S 3      L 1
Program    PGM01      Library SYSTEM
0120      DEFINE DATA
0130      LOCAL
0140      01 #NAME-START      (A20)
0150      01 #NAME-END        (A20)
0160      01 #MARK            (A1)
0170      01 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    
```

Each of the three variables has a "#" as the initial character. This character is used to identify user-defined variables; that is, to distinguish them from database fields.

Enter the command CHECK to see if everything you typed in as part of the new local data area is correct. This command ends split-screen mode and you return to the full-screen mode.

Step 4

The local data area is not yet complete. You can read the other variables you need directly from a view into the data area editor.

In the line below the variables you have already defined, enter the command ".V (EMPLOYEES)" (starting in column "T"). The view EMPLOYEES will be displayed:

Local	Library	SYSTEM	DBID	10	FNR	32
View	EMPLOYEES					
I T L	Name	F	Leng	Index/Init/EM/Name/Comment		

	2 PERSONNEL-ID	A	8			
G	2 FULL-NAME					
	3 FIRST-NAME	A	20			
	3 MIDDLE-I	A	1			
	3 NAME	A	20			
	2 MIDDLE-NAME	A	20			
	2 MAR-STAT	A	1			
	2 SEX	A	1			
	2 BIRTH	D				
	2 NJBIRTH	I	2			
G	2 FULL-ADDRESS					
M	3 ADDRESS-LINE	A	20	(1:191) /* MU-FIELD		
	3 CITY	A	20			
	3 ZIP	A	10			
	3 POST-CODE	A	10			
	3 COUNTRY	A	3			
G	2 TELEPHONE					

SYSGDA 4461: Mark fields to incorporate into data area.						

Step 5

From this view (DDM), select the fields which you want to include in the local data area by marking them with any character in column "I". In this case, mark the fields PERSONNEL-ID, NAME, DEPT and LEAVE-DUE. DEPT and LEAVE-DUE are not in the first section of the view; to scroll forward within the view, keep pressing ENTER until the section which contains DEPT and LEAVE-DUE is displayed.

After you have marked all the fields indicated, continue pressing ENTER until the local data area - which now includes the fields you have selected from the EMPLOYEES view - is displayed again:

```

Local                               Library SYSTEM                               DBID  10 FNR  32
Command
I T L Name                          F Leng Index/Init/EM/Name/Comment
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
   2 NAME                              A   20
   2 DEPT                              A    6
   2 LEAVE-DUE                         N   2.0

----- S 8      L 1

SYSGDA 4462: 4 field(s) of view EMPLOYEES included.

```

Step 6

The local data area is now complete. To check if it contains any errors, enter the command CHECK in the command line of the editor.

If it contains any errors, correct them and repeat the CHECK.

Then enter the command STOW LDA01 in the command line. The local data area is now compiled and stored in source and object form under the name LDA01.

Note that a data area must be compiled and stored in object form before any program referencing that data area can be compiled.

Step 7

Now program PGM01 must be edited to reference the local data area LDA01.

In the command line of the data area editor, enter the command EDIT PGM01. This invokes the program editor and reads the program PGM01 into the work area of the program editor.

Step 8

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

Enter the command ".D" at the beginning of each line that contains a variable definition within the DEFINE DATA statement, as shown below:

```

>
> + Program      PGM01      Lib SYSTEM
....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
0010 * Example Program: PGM01
0020 * Function: Demonstrate Natural
0030 * -----
0040 DEFINE DATA
0050     LOCAL
0060 .d 01 #NAME-START          (A20)
0070 .d 01 #NAME-END          (A20)
0080 .d 01 #MARK              (A1)
0090 .d 01 EMPLOYEES-VIEW VIEW OF EMPLOYEES
0100 .d 02 PERSONNEL-ID       (A8)
0110 .d 02 NAME               (A20)
0120 .d 02 DEPT              (A6)
0130 .d 02 LEAVE-DUE         (N2)
0140 END-DEFINE

```

When you press ENTER, these lines are deleted from the program.

Then enter the following after the word LOCAL:

```
USING LDA01
```

The program should now look as follows:

Program PGM01:

```
* Example Program 'PGM01' for User's Guide Tutorial
* -----
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
    THRU #NAME-END
  IF LEAVE-DUE >= 20
    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
    REINPUT 'PLEASE TRY ANOTHER NAME'
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
*
END
```

Step 9

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 and a blank (*) 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.

In the second line of the program, enter the command ".I(1)" to insert an empty line. In the empty line, add some comment to indicate that this program has been updated, as shown in the example below.

Example:

```
* Example Program 'PGM01' for User's Guide Tutorial* PROGRAM NOW USES A LOCAL DATA AREA
* -----
DEFINE DATA
  LOCAL USING LDA01 /* This comment is for demonstration purposes only.
END-DEFINE
...
```

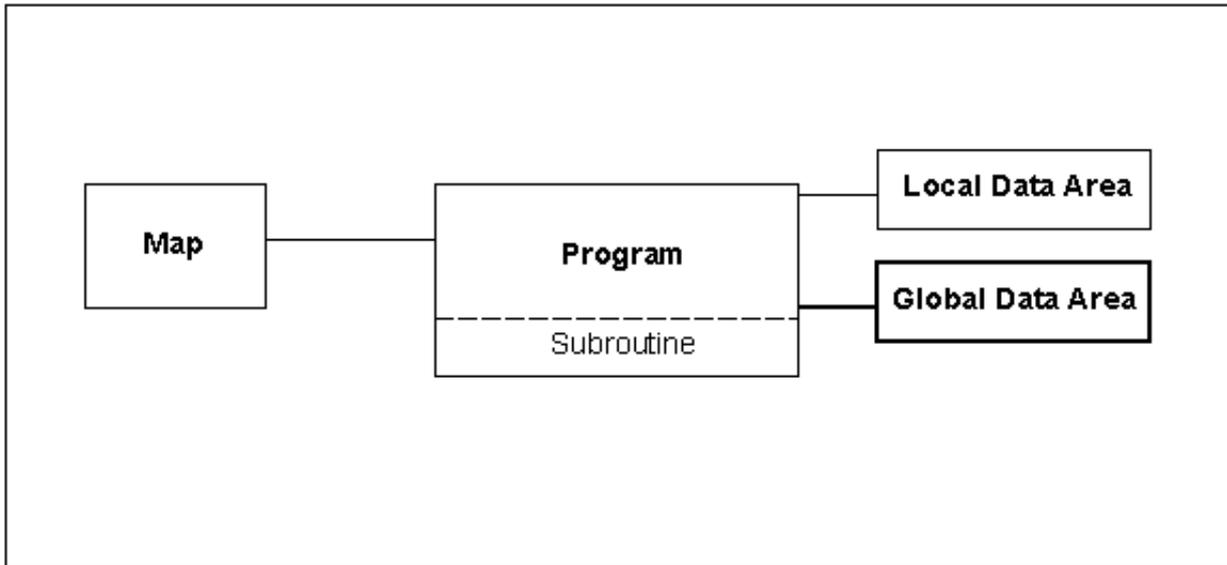
The subsequent sessions in this section show further examples of comments, which you can add to keep track of changes you have made.

Step 10

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 create a global data area. In addition, the local data area created in Session 2 will be modified. Moreover, the program has to be modified to reference not only the local data area, but also the new global data area.

Step 1

On the Development Functions menu, enter the code "E" and name LDA01. This invokes the data area editor and reads the local data area LDA01 into the work area of the editor.

Step 2

To create the new data area, save the contents of the work area under a different name: in the command line of the editor, enter the command `SAVE GDA01`.

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

Step 3

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*.

Enter the line command ".D" to delete the first two lines (#NAME-START and #NAME-END). The data area now looks as follows:

Global	GDA01	Library	SYSTEM	DBID	10	FNR	32	
Command							> +	
I	T	L	Name	F	Leng	Index/Init/EM/Name/Comment		
All	-	-----					-	-----
	1		#MARK	A	1			
V	1		EMPLOYEES-VIEW			EMPLOYEES		
	2		PERSONNEL-ID	A	8			
	2		NAME	A	20			
	2		DEPT	A	6			
	2		LEAVE-DUE	N	2.0			
-----							S 6	L 1

Keep in mind that a data area must be compiled and stored in object form before any program referencing that data area can be compiled and executed.

In the command line, enter the command STOW. GDA01 is now compiled and stored 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.

Enter the command READ LDA01 in the command line to read the local data area LDA01 into the editor.

With the line command ".D", 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 data area now looks as follows:

Local LDA01		Library SYSTEM	DBID 10 FNR 32	
Command			>+	
I T L	Name	F	Leng	Index/Init/EM/Name/Comment
All	-	-	-	-
1	#NAME-START	A	20	
1	#NAME-END	A	20	
				S 2 L 1

In the command line, enter the command STOW.

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

Step 5

In the command line of the data area editor, enter the command EDIT PGM01. As PGM01 is stored as an object of type program, the program editor will automatically be invoked.

As the defined data are now located in two data areas, the DEFINE DATA statement in the program must now reference the global data area GDA01 as well as the local data area LDA01.

In the line which contains DEFINE DATA, enter the command ".I(1)" to insert one empty line. Type in GLOBAL USING GDA01 in the empty line.

The DEFINE DATA statement block should now look as follows:

```
...  
DEFINE DATA  
  GLOBAL USING GDA01  
  LOCAL  USING LDA01  
END-DEFINE  
...
```

Step 6

Moreover, we will change the instructions for the output produced by the program: we will add a WRITE TITLE statement and modify the DISPLAY statement.

Using the program example below, modify the program to include the WRITE TITLE statement (above the DISPLAY statement) and the new format in the DISPLAY statement. Use the line command ".I" 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 WRITE TITLE statement used in this program produces a multiple line title 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.

The revised program - particularly the DEFINE DATA, WRITE TITLE and DISPLAY statements blocks - should now look as follows:

Program PGM01:

```

* Example Program 'PGM01' for User's Guide Tutorial
* PROGRAM NOW USES A LOCAL DATA AREA
* A GLOBAL DATA AREA AND TITLE HAVE BEEN ADDED AND
* THE 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 >= 20
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
WRITE TITLE / '*** PERSONS WITH 20 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 completed all changes, CHECK the program and correct any errors that may have occurred.

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

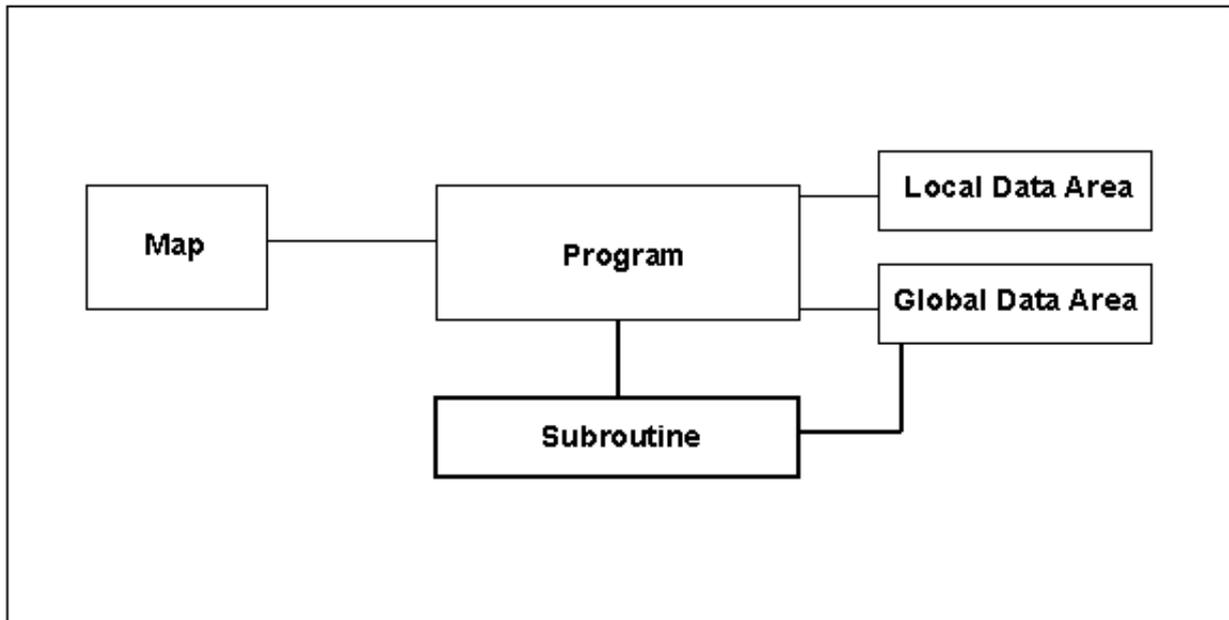
Note the differences in the report output, which should now look as follows:

*** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***			
*** ARE MARKED WITH AN ASTERISK ***			
N A M E	DEPT	LV	*
-----	-----	---	-
JONES	SALE30	25	*
JONES	MGMT10	34	*
JONES	TECH10	11	
JONES	MGMT10	18	
JONES	TECH10	21	*
JONES	SALE00	30	*
JONES	SALE20	14	
JONES	COMP12	26	*
JONES	TECH02	25	*

When the execution of the program has finished without any errors, STOW the program for future modification in Session 4.

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

If program PGM01 is not already in the program editor, enter the code "E" and name PGM01 on the Development Functions menu.

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

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*.

Step 2

Delete all lines of the subroutine except the comment lines, the DEFINE DATA and DEFINE SUBROUTINE blocks, and the END statement, so that the program looks like the illustration below.

You can delete lines quickly by marking the first line of a block of lines with the line command ".X" and the last line of a block with ".Y" and then entering the command "DX-Y" in the command line to delete the specified block. Add a comment to identify this subroutine.

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 3

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

In the command line of the editor, enter READ PGM01.

Step 4

Enter the command BOTTOM in the command line to move to the subroutine definition at the end of the program. Delete the following lines, containing the internal subroutine definition, from the program:

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

The program should now look like the example below:

Program PGM01

```

* Example Program 'PGM01' for User's Guide Tutorial
* PROGRAM NOW USES A LOCAL DATA AREA
* A GLOBAL DATA AREA AND TITLE HAVE BEEN ADDED AND
* THE DISPLAY STATEMENT HAS BEEN CHANGED
* THE SUBROUTINE IS NOW EXTERNAL
* -----
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 >= 20
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
  WRITE TITLE / '*** PERSONS WITH 20 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

```

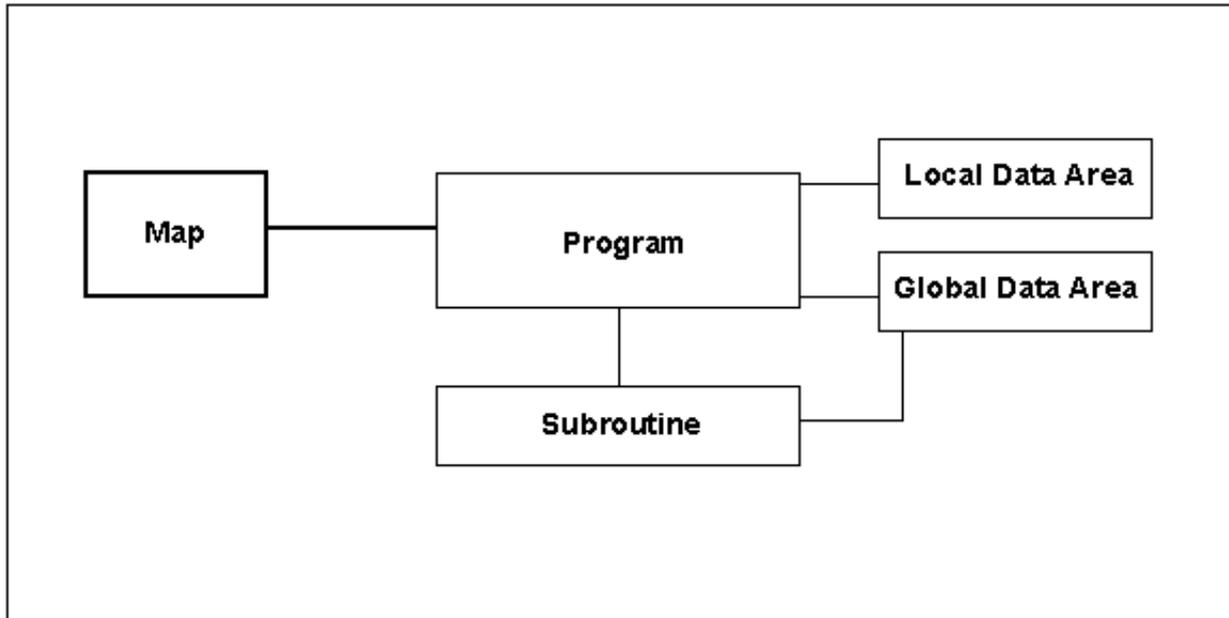
Step 5

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.

End of Session 4.

Session 5 - Editing a Map



In the previous sessions, the screen (map) prompting for an employee name was invoked via the INPUT USING MAP statement. In this session, the map - MAP01 - will be edited to add an ending name for a range of employees and some new layout elements.

The Natural *map editor* is used to create and modify screen layouts quickly and efficiently. In this session, you will use several map editor line commands and field commands:

- A line command begins with two periods (..). You enter it at the beginning of a line, and it applies to the whole line in which you enter it.
- A field command begins with one period (.). You enter it at the beginning of a field, and it applies only to the field in which you enter it.

(The sessions in the section Tutorial - Using the Map Editor also show you how to apply these commands to more than one line/field at a time.)

For detailed descriptions of all map editor commands, refer to Editing a Map in the section Map Editor.

Step 1

To invoke the map editor, enter the code "E" and type "M" on the Development Functions menu. The Edit Map menu will be displayed:

```

16:51:35          ***** NATURAL MAP EDITOR *****          2001-01-31
User SAG          - Edit Map -          Library SYSTEM

          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

```

The map editor provides an extensive *help system*, which you can invoke by entering a question mark (?) as code on the Edit Map menu. Take time to look through this help system so that you know what kind of help is available.

Step 2

To edit map MAP01, enter the code "E" and name MAP01 on the Edit Map menu. The editing screen of the map editor will be displayed:

```

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---+-----

                Please enter starting name :XXXXXXXXXXXXXXXXXXXXX (. to exit)

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

The editing screen will be displayed in split-screen mode. The top portion of the screen can be used, for example, to display view definitions; the upper right corner displays delimiter settings that apply for the map. The lower portion of the screen is the map editing area.

Map fields may be defined directly on the screen or they can be selected from a view (DDM) that is displayed in the upper portion of the screen. In this exercise, map fields will be defined directly on the screen.

Unlike the program editor and the data area editor, the map editor does not have a command line. Many functions in the map editor are performed by using PF keys (the PF-key lines at the bottom of the screen show which function is assigned to which key).

The two screens below show the map editor screen for map MAP01 as it will appear at the end of this session, and the display of the screen as it will appear when MAP01 is invoked while executing program PGM01.

```

Fld #NAME-END                                     Fmt A20
-----
AD= MIT'_'_____      ZP= OFF      SG= OFF      HE= _____      Rls 0
AL= _____          CD=  ___      CV= _____          Mod User
PM=  __  DF=           DY= _____
EM= _____

001  --010---+-----+-----030---+-----+-----050---+-----+-----070---+-----
(XXXXXXXXX                Software AG Employee Information                (XXXXXXXXX
(XXXXXXXXXXXXX

                Please enter starting name :XXXXXXXXXXXXXXXXXXXXX (. to exit)
                                Ending name .XXXXXXXXXXXXXXXXXXXXX

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

```

31/01/01                Software AG Employee Information                SYSTEM
16:01:26.8

                Please enter starting name _____ (. to exit)
                                Ending name _____
    
```

Step 3

In the first line of the editing area, type in the following text, as shown below:

Software AG Employee Information

Then insert two blank lines by entering the line command ".I(2)" in the first six positions of the text you just typed in. The map now looks 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---+-----
Software AG Employee Information

Please enter starting name :XXXXXXXXXXXXXXXXXXXXX (. to exit)
    
```

Step 4

Enter the line command ".C" in the first three positions of the "Software AG Employee Information" line. The text is now centered:

```

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---+-----
                Software AG Employee Information

Please enter starting name :XXXXXXXXXXXXXXXXXXXXX (. to exit)
    
```

Step 5

Enter (*DATE, (*TIME and (*LIBRARY-ID as shown below:

```

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---+---
(*DATE                               Software AG Employee Information      (*LIBRARY-ID
(*TIME

Please enter starting name :XXXXXXXXXXXXXXXXXXXXX (. to exit)
    
```

*DATE, *TIME, and *LIBRARY-ID are Natural system variables. System variables all begin with an asterisk (*). When the program which invokes the map is executed, *DATE will display the current date, *TIME the current time, and *LIBRARY-ID your current library. For more information on system variables, see the section System Variables in the Natural Programming Reference documentation.

The opening parenthesis "(" in front of the system variable is a *delimiter* character. A delimiter indicates the combination of class and attribute assigned to a field. In this case, the "(" delimiter identifies the fields as a non-modifiable, intensified, output-only fields. The currently valid delimiter characters are shown in the top right-hand corner of the map editing screen.

Class types (column CLS) shown on this screen include:

Class Type	Description
T	Text constant
A	Input field
O	Output-only field (non-modifiable)
M	Modifiable field (output and input field)

Attribute types (column ATT) shown on this screen include:

Attribute Type	Description
D	Default (that is, non-intensified, non-blinking, etc.)
I	Intensified

By entering these delimiter characters directly in front of a field, you assign a class and attribute to that field. Other class and attribute combinations are possible. It is also possible to assign another delimiter character to a specific class/attribute combination.

After you have typed in the system variables along with the delimiter characters, press ENTER. The system variable names entered will be transformed on the map to a series of Xs:

```

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---+-----
(XXXXXXXXX          Software AG Employee Information          (XXXXXXXXX
(XXXXXXXXXXXX

Please enter starting name :XXXXXXXXXXXXXXXXXXXXXXXXX (. to exit)
    
```

Step 6

Add another field to the map. Type in as shown in the screen below:

Ending name :X(20)

The colon (:) indicates that the field is modifiable for user input, and is displayed intensified.

```

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---+-----
(XXXXXXXXX          Software AG Employee Information          (XXXXXXXXX
(XXXXXXXXXXXXX

Please enter starting name :XXXXXXXXXXXXXXXXXXXXXXXXX (. to exit)
Ending name :x(20)
    
```

Press ENTER, and the new field is added, its length determined by the 20 Xs:

```

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---+-----
(XXXXXXXXX          Software AG Employee Information          (XXXXXXXXX
(XXXXXXXXXXXXX

Please enter starting name :XXXXXXXXXXXXXXXXXXXXXXXXX (. to exit)
Ending name :XXXXXXXXXXXXXXXXXXXXXXXXX
    
```

Step 7

This newly created field needs further definition before it can be processed in the program PGM01.

Enter the field command ".E" in the first two positions of the field , as shown below:

```

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---+---
(XXXXXXXXX                Software AG Employee Information          (XXXXXXXXX
(XXXXXXXXXXXXX

Please enter starting name :XXXXXXXXXXXXXXXXXXXXXXXXX (. to exit)
Ending name .XXXXXXXXXXXXXXXXXXXXXXXXX
    
```

This causes the extended field editing section to be invoked for the field you have marked with the command. (Another way to invoke extended field editing function is to position the cursor anywhere in the field and press PF5.)

The extended field editing section will be displayed in the top half of the screen:

```

Fld #001                                                                    Fmt A20
-----
AD= MIT'_'_____ ZP= OFF      SG= OFF      HE= _____ Rls 0
AL= _____    CD=  _      CV= _____ Mod Undef
PM=  _  DF=      DY= _____
EM= _____

001  --010---+-----+---030---+-----+---050---+-----+---070---+---
(XXXXXXXXX                Software AG Employee Information          (XXXXXXXXX
(XXXXXXXXXXXXX

Please enter starting name :XXXXXXXXXXXXXXXXXXXXXXXXX (. to exit)
Ending name .XXXXXXXXXXXXXXXXXXXXXXXXX
    
```

The field "Fld" displays as field name a number (#001). Natural automatically assign such a number to every new field that is created in a map.

Step 10

To leave the Edit Map menu, enter the code (.), and you will be returned to the Development Functions menu.

Enter the command E PGM01 in the command line. The program editor will be invoked and program PGM01 will be read into the work area ready to be adjusted to the modified map.

Step 11

Until now, PGM01 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 JONES. (Otherwise, all employees from JONES to the end of the alphabet would have been included in the report.) Now the map allows us to specify both a start value **and** an end value for the list to be output. However, an IF statement must be added to the program to handle a situation in which no end value is specified.

Change the program by inserting the following lines before the READ statement:

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

Add comments to reflect program changes.

The program should now look as follows:

Program PGM01:

```

* Example Program 'PGM01' for User's Guide Tutorial
* PROGRAM NOW USES A LOCAL DATA AREA
* A GLOBAL DATA AREA AND TITLE HAVE BEEN ADDED AND
* THE DISPLAY STATEMENT HAS BEEN CHANGED
* THE SUBROUTINE IS NOW EXTERNAL
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
* -----
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
        THRU #NAME-END
        IF LEAVE-DUE >= 20
          PERFORM MARK-SPECIAL-EMPLOYEES
        ELSE
          RESET #MARK
        END-IF
*
WRITE TITLE / '*** PERSONS WITH 20 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 12

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

Then RUN the program. On the input screen, enter the names JONES and JOY as start value and end value respectively. The following output report will appear:

```

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

```

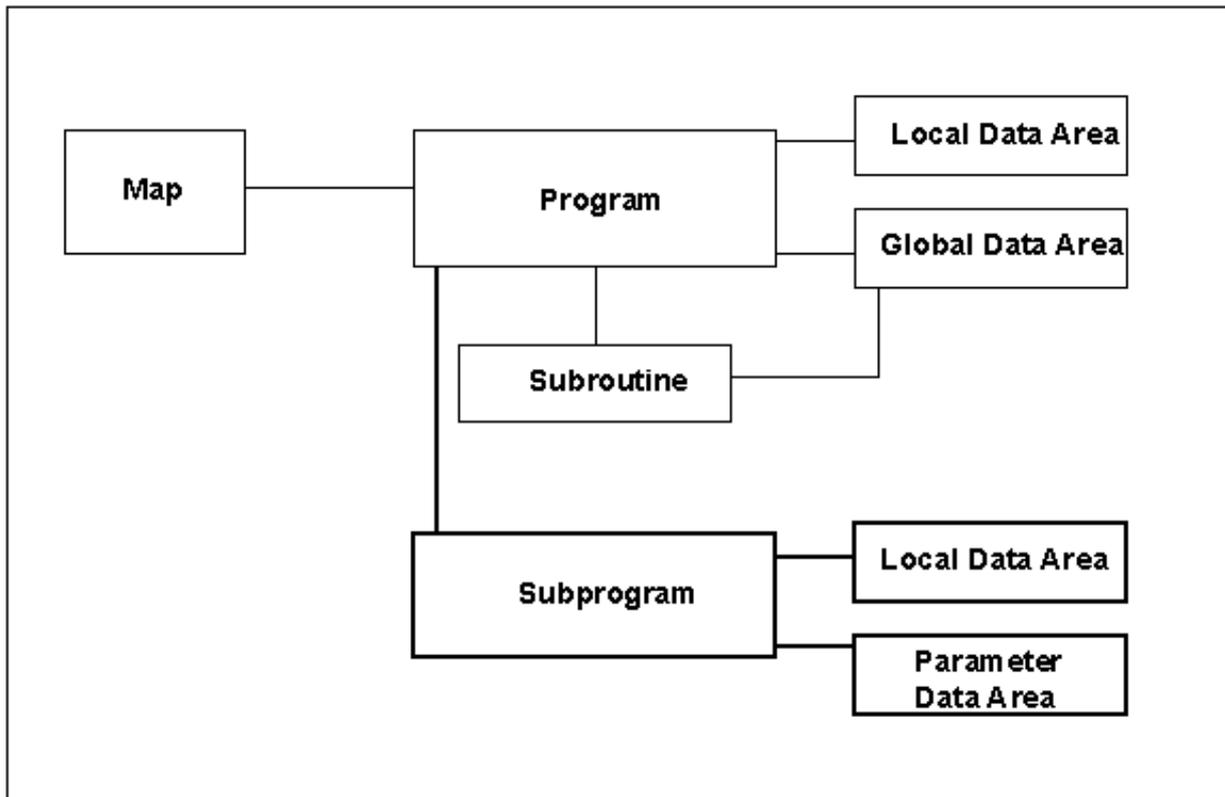
N A M E	DEPT	LV	*
-----	-----	---	-
		DUE	
JONES	SALE30	25	*
JONES	MGMT10	34	*
JONES	TECH10	11	
JONES	MGMT10	18	
JONES	TECH10	21	*
JONES	SALE00	30	*
JONES	SALE20	14	
JONES	COMP12	26	*
JONES	TECH02	25	*
JOPER	MARK29	32	*
JOUSSELIN	FINA01	45	*

STOW the program for future use.

To return to the Development Functions menu, enter a period (.) in the command line of the editor.

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 employees information.

Step 1

First, the main program's local data area, LDA01, must be modified.

On the Development Functions menu, enter the code "E" (Edit Object) and name LDA01. The data area editor will be invoked, and the local data area LDA01 will be read into the editing area. LDA01 should appear as follows:

Local	LDA01	Library	SYSTEM	DBID	10	FNR	32
Command							> +
I T L	Name			F	Leng	Index/Init/EM/Name/Comment	
All	-	-----	-	----	-----	-----	-----
	1	#NAME-START		A	20		
	1	#NAME-END		A	20		

Step 2

Add the variables #PERS-ID, #MAKE and #MODEL to the local data area, as show below. They will be referenced in the CALLNAT statement to be added to the program. The local data area now looks as follows:

Local	LDA01	Library	SYSTEM	DBID	10	FNR	32
Command							> +
I T L	Name			F	Leng	Index/Init/EM/Name/Comment	
All	-	-----	-	----	-----	-----	-----
	1	#NAME-START		A	20		
	1	#NAME-END		A	20		
	1	#PERS-ID		A	8		
	1	#MAKE		A	20		
	1	#MODEL		A	20		

CHECK and STOW the local data area.

Step 3

With minor modifications it is possible to use the local data area LDA01 to create the parameter data area that will be needed for the subprogram. (Instead of creating a separate parameter data area, it would also be possible to define the parameter data directly within the subprogram's DEFINE DATA PARAMETER statement.)

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.

Then enter the command SET TYPE PARAMETER to change the data area type from Local to Parameter.

Step 4

With the line command ".D", delete the first two lines:

```
1 #NAME-START A 20
1 #NAME-END   A 20
```

The parameter data area now looks as follows:

Parameter	PDA02	Library	SYSTEM	DBID	10	FNR	32
Command							> +
I T L Name			F Leng	Index/Init/EM/Name/Comment			
All	-----						
1	#PERS-ID		A	8			
1	#MAKE		A	20			
1	#MODEL		A	20			

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

Step 5

Like a program, a subprogram can have its own local data area. This will be created now. First enter the command CLEAR to clear the contents of the editing area. Then change the data area type to Local with the command SET TYPE LOCAL.

Step 6

In the first line of the editing area, enter the line command ".V (VEHICLES)" starting in column "T". The view VEHICLES will be displayed listing all fields contained in that view. Select the following fields to be included into the data area: PERSONNEL-ID, MAKE, and MODEL (see also Session 2, Step 5). These fields will be automatically incorporated into the local data area.

The new local data area should now appear as follows:

Local	Library		SYSTEM	DBID	10	FNR	32
Command							> +
I T L Name			F Leng	Index/Init/EM/Name/Comment			
All	-----						
V 1	VEHICLES-VIEW			VEHICLES			
2	PERSONNEL-ID		A	8			
2	MAKE		A	20			
2	MODEL		A	20			

Step 7

Enter the command SAVE LDA02 to store the data area in source form. Then CHECK the data area, and correct any errors if necessary. Then STOW the data area to compile it and store it in source and object form.

The local data area is now ready to be used by the subprogram.

Step 8

The next step is to create the subprogram itself.

Leave the data area editor by entering a period (.) in the command line. On the Development Functions menu, enter the code "E" (for Edit Object) and type "N" (for subprogram). The program editor will be invoked with an empty work area.

Type in the subprogram as shown below. Then SAVE it under the name SPGM02 (SAVE SPGM02). CHECK it and correct any errors, and then STOW it.

```

Subprogram SPGM02:
* Example Subprogram 'SPGM02'
* *****
DEFINE DATA
  PARAMETER USING PDA02
  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 receives the personnel number passed by the main program and uses this number as the basis of a search of the VEHICLES file.

Step 9

Now the main program must be modified to invoke the subprogram.

Read the program into the editor with the command READ PGM01, 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. Note also 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.

Step 10

As the program receives vehicle information from the subprogram, the DISPLAY statement must be expanded as follows:

Previous DISPLAY Statement:

```
DISPLAY 23X '//NAME' NAME
        3X '//DEPT' DEPT
        3X '//LV/DUE' LEAVE-DUE
        3X '//*' #MARK
```

Expanded DISPLAY Statement:

```
DISPLAY '//NAME' NAME
        2X '//DEPT' DEPT
        2X '//LV-/DUE' LEAVE-DUE
        ' ' #MARK
        2X '//MAKE' #MAKE
        2X '//MODEL' #MODEL
```

After you have made all modifications, the program should look as follows:

Program PGM01:

```

* Example Program 'PGM01' for User's Guide Tutorial
* PROGRAM NOW USES A LOCAL DATA AREA
* A GLOBAL DATA AREA AND TITLE HAVE BEEN ADDED AND
* THE DISPLAY STATEMENT HAS BEEN CHANGED
* THE SUBROUTINE IS NOW EXTERNAL
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
* 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 EMPLOYEES-VIEW
    BY NAME
    STARTING FROM #NAME-START
    THRU #NAME-END
*
  IF LEAVE-DUE >= 20
    PERFORM MARK-SPECIAL-EMPLOYEES
  ELSE
    RESET #MARK
  END-IF
*
  RESET #MAKE #MODEL
  CALLNAT 'SPGM02' PERSONNEL-ID #MAKE #MODEL
WRITE TITLE / '*** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***'
           / '*** ARE MARKED WITH AN ASTERISK ***'//
*
  DISPLAY   '//N A M E' 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 11

Once the program modifications have been made, CHECK the program and correct any errors.

Then RUN the program. Enter JONES and JOY as starting and ending names on the input screen. The report produced by the program now looks as follows:

*** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***				
*** ARE MARKED WITH AN ASTERISK ***				
N A M E	DEPT	LV 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
JOPER	MARK29	32		*
JOUSSELIN	FINA01	45	RENAULT	* R25

After the program has been executed, STOW it for future reference.

End of Session 6.