

Tutorial - Getting Started with Natural

This tutorial is designed to provide a basic understanding of specific features of the Natural programming environment and illustrates how an application can be structured as a group of modules. It is **not** intended to provide an example of how an application should be built.

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

Program Editor | Data Area Editor | Map Editor | DDM Editor | Dialog Editor

Prerequisite:

To perform all steps of this tutorial, the database SAG-DEMO-DB must be installed and active. To start the database, double click the SAG-DEMO-DB icon in the Natural program group.

- Session 1 - Creating and Modifying a Program
 - Session 2 - Creating and Editing a Map
 - Session 3 - Checking and Running a Program
 - Session 4 - Creating a Local Data Area
 - Session 5 - Creating a Global Data Area
 - Session 6 - Creating an External Subroutine
 - Session 7 - Invoking a Subprogram
-

Session 1 - Creating and Modifying a Program

In this session, you will create and save a Natural program, using the program editor to enter source statements in a program editor window.

Step 1

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 application created in these sessions will be stored in the SYSEXPG library.

Select the SYSEXPG library node in the library workspace.

To open the library SYSEXPG

1. From the tree view, choose "System Libraries".
2. Scroll to SYSEXPG and select it.

Step 2

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

Software AG recommends that you use structured mode exclusively, because it results in more clearly structured applications. Therefore all explanations and examples in this chapter refer to structured mode. Any properties of reporting mode will not be taken into consideration. You must be operating in structured mode to work through the sessions in this chapter.

If the current mode is reporting mode, change it to structured mode:

To do so

1. From the **Tools** menu, choose **Session Parameters > Compiler options**.
2. Select "Structured Mode".
3. Choose **OK**.

Step 3

The SYSEXPG library should include the program used in this session, PGM01. In this step, you will either edit or create the program.

Edit PGM01

If PGM01 is available, edit the program.

To open PGM01 for editing

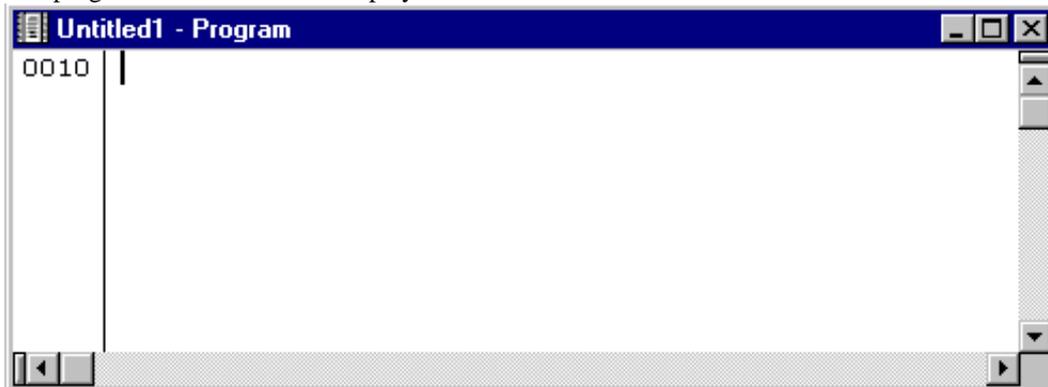
- Expand the library node, expand the "Programs" node, select the Program PGM01 and press ENTER.

Create PGM01

If PGM01 is not available, you can create it.

▶ To open a new program editor window

- Open the context menu of the "Programs" node and select the **New** item.
The program editor window is displayed:



▶ To modify "Program Editor Options"

If line numbers are not usable, you can modify them.

1. From the main menu bar select **Tools > Options > Program Editor**.
2. Set "Line Numbers" check box.

Step 4

▶ To save the program under the name "PGM01"

1. From the **Object** menu, choose **Save**.
If the program already exists in the library, then it is saved. Go to Step 5.
If the program does not yet exist in the library, the "Save as" dialog box appears.
2. In the "Name" text box, enter "PGM01".
3. Choose **OK**.

The program is now saved under the name "PGM01" in the library SYSEXPG.

Step 5

▶ To close PGM01 before ending the session

- From the **Object** menu, select **Close** or press **CTRL-F4**.

End of Session 1.

Session 2 - Creating and Editing a Map

The Natural map editor is used for creating the maps referenced in a Natural program. Once a map has been created, it can be stored in the Natural system file, where it can be invoked by a Natural program using a WRITE or INPUT statement.

A map consists of fields. A field can be a text field (a constant) or a data field (a variable), or any of the graphical user interface elements provided in the map editor's "Insert" menu. The fields that comprise a map can be defined direct in the map editor window, or imported from another source object, such as a DDM, a program, or a data area. Natural system variables can be imported as well.

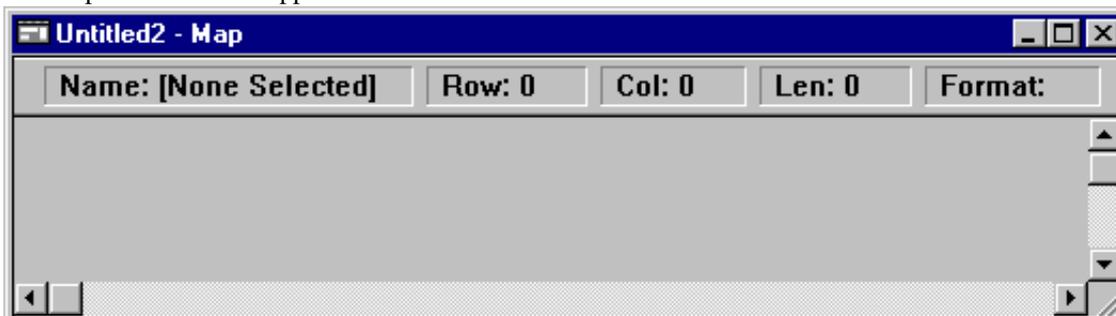
In this session, you will create a map that contains text fields, data fields, and system variables.

Step 1

In the previous session, the screen prompting for an employee name was produced through the INPUT USING MAP statement using MAP01. In this session, you will create the map. Note that in the INPUT USING MAP statement, the map must be specified in quotation marks to distinguish the map from a user-defined variable.

▶ To open a new map editor window

- Open the context menu of the SYSEX EVT node and select the **New > Map** item. The map editor window appears.



Step 2

A text field is a constant that you create using the text field entry in the **Insert** menu, or that you import from another Natural object. Its format is always A (for alphanumeric).

You can create a title for the map by drawing a text field and defining the text it will contain.

▶ To do so

1. From the **Insert** menu, choose **Text Constant**.
Or click the **Text Constant** toolbar button.
2. Place the text field at the top of the editor, where you want the field to begin.
3. Draw a field by holding down the left mouse button and dragging the mouse to the right about half the width of the editor.
The text field you have just drawn is still selected. When a field is selected, its field handles appear.

The field must be selected before you can perform many of the map editor functions, such as defining a field and selecting a color for the field.

▶ To define the text field

1. Point to the field and double-click.
Or, from the **Field** menu, choose **Definition**.
In the text field, you can now enter the text.
2. Type "SOFTWARE AG EMPLOYEE INFORMATION".
Select the field again by clicking the mouse with the pointer outside the field and then with the pointer on the field.

▶ To select a color for the text field

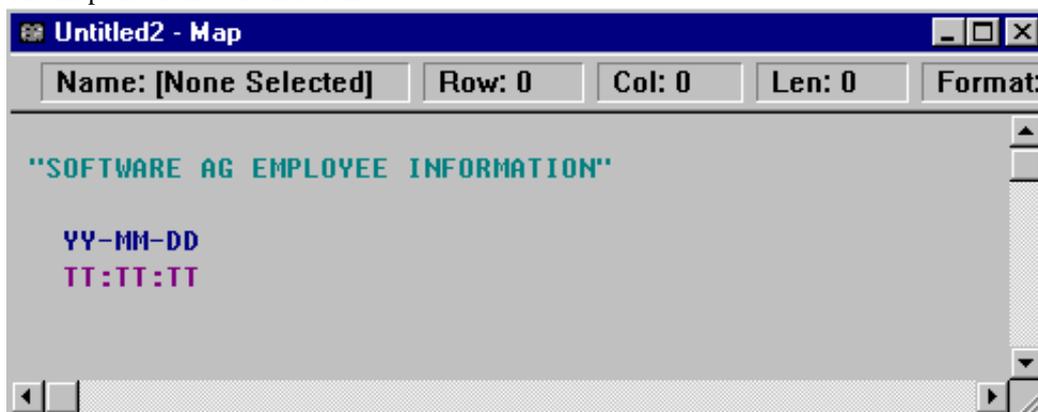
1. From the **Field** menu, choose **Color**.
Or click the **Field Color** toolbar button.
2. Select any color you want for this field. (Click the name of the color or its **Option** button.)
3. Choose **OK**.
"SOFTWARE AG EMPLOYEE INFORMATION" appears on the map in the color you selected.
4. To deselect the field, move the pointer away from the field and click.
The field handles disappear.

Step 3

Natural system variables can be imported into a map. The system variables *DATX and *TIMX display the current date and time, when the program that invokes the map is executed.

▶ To import the *DATX system variable

1. From the **Insert** menu, choose **Import**.
2. Choose **System variable**. The "Import System Variable" dialog box appears.
3. Scroll to *DATX and select it.
4. Choose **Import**. The system variable will appear in the top left corner of the map.
5. Choose **Quit** to close the dialog box.
6. Move the *DATX field cursor below SOFTWARE AG EMPLOYEE INFORMATION.
7. Select a color for the *DATX field.
Import the *TIMX system variable. Use the same procedure you used to import the *DATX system variable.
Select a color for the *TIMX field, then move *TIMX to the line below *DATX.
The map should now look as follows.



Step 4

New fields can be created by copying and redefining existing fields.

▶ To copy a field to the clipboard

1. Select the text field "SOFTWARE AG EMPLOYEE INFORMATION".
2. From the **Edit** menu, choose **Copy**.

▶ **To paste the copied field into the map**

1. From the **Edit** menu, choose **Paste**.
2. Drag the copied field from the top left corner to below the *TIMX field.
Notice that this field is the same color as the field you copied. If you want to change its color, from the **Field** menu, select **Color**.

▶ **To define the new field in the "Text Field Definition" dialog box**

1. Point to the field and double-click.
Or, from the **Field** menu, choose **Definition**.
In the text field, you can now enter the text.
2. Type "PLEASE ENTER STARTING NAME:".

Step 5

A data field is a field that you create using the Data field entry in the "Insert" menu, a field that you import from another Natural object, or a Natural system variable.

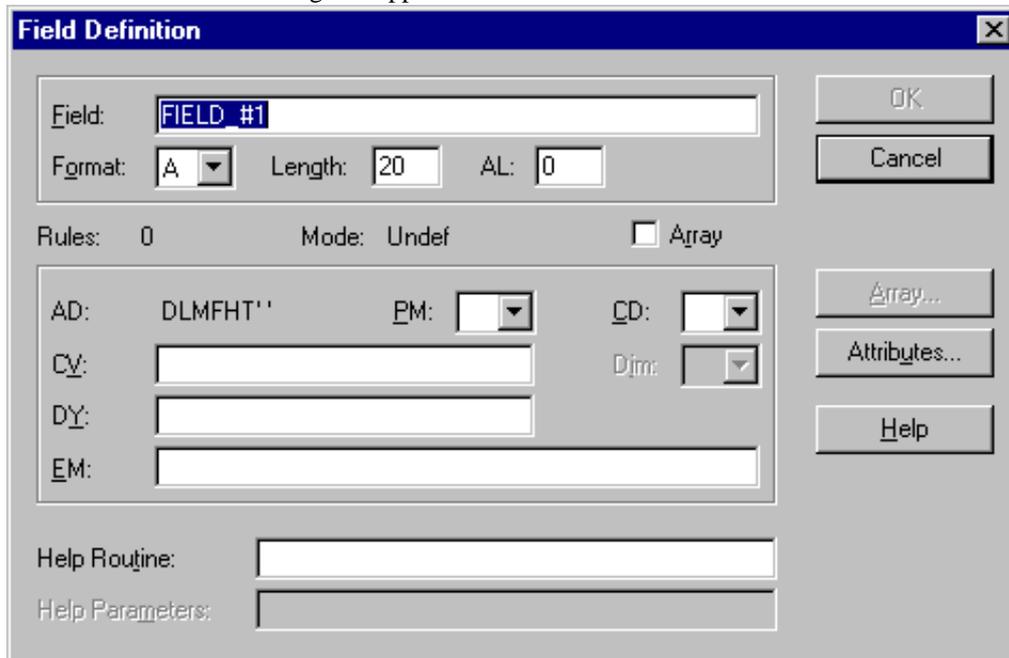
In this step, you will draw a data field and define its attributes.

▶ **To draw the data field**

1. From the **Insert** menu, choose **Data Field**.
Or click on the **data field drawing tool** toolbar button.
2. Place the data field to the right of PLEASE ENTER STARTING NAME:
3. Draw a field that is 20 characters long. (Using the mouse, drag the data field across the map until Len=20).

▶ **To define the data field**

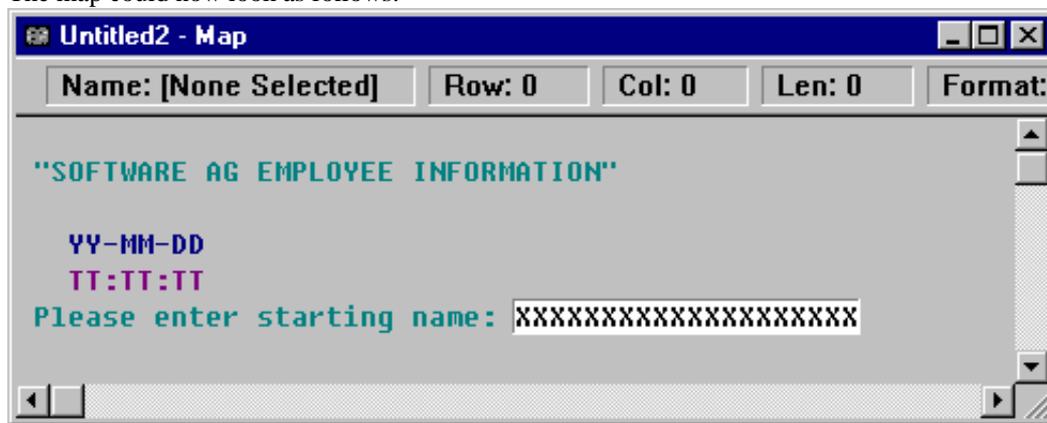
1. Point to the field and double-click.
Or from the **Field** menu, choose **Definition**.
The "Field Definition" dialog box appears:



2. In the "Field" text box, delete the name and type in "#NAME-START". Press the TAB key.
Format "A" (alphanumeric) is the correct format for this field.
The alphanumeric length of the field should be "20". If not, use TAB to move the cursor to the "Length" field and enter "20".

▶ **To specify attributes for the data field**

1. Choose **Attributes**.
The "Attribute Definitions" dialog box appears.
2. Select the "I/O Characteristics" list box and select "Output, Modifiable" to define the field as an output field that can be modified.
3. Enter underscore () as filler character.
This is the character that is used to fill any empty positions in input fields in the map, allowing the user to see the exact position and length of a field when entering input.
4. Choose **OK**.
The "Field Definition" dialog box is displayed again.
5. Choose **OK** to save the data field definition that you entered.
The map could now look as follows:



Step 6

In this step, you will edit the map to add an ending name for a range of employees.

In the same way as you have created text fields and data fields so far, draw and define another text field and another data field.

To draw and define the "PLEASE ENTER ENDING NAME:" text field

1. Choose **Insert > Text Constant** to create the text field and draw a field 25 characters long, one line below "PLEASE ENTER STARTING NAME:"
2. In the text field, enter PLEASE ENTER ENDING NAME:
3. Select a color for the text field.

To draw and define the data field "#NAME-END"

1. Choose **Insert > Data Field** to draw a field 20 characters long, one space to the right of the text constant.
2. In the "Data Field Definition" dialog box, enter "#NAME-END" as the field name (the format is "A" and the length is "20").
3. Choose **Attributes** and select "Output, Modifiable" as the I/O Characteristic.
4. Choose **OK** twice.
5. Select a color for the "#NAME-END" data field.

The output of this data field is a user-defined variable found in the DEFINE DATA statement of PGM01 that will correspond to the new field definition entered on the map.

Step 7

To center the field "SOFTWARE AG EMPLOYEE INFORMATION" at the top of the map

1. Select the field.
2. From the **Field** menu, choose **Alignment**.
3. From the cascading menu, choose **Map center**.
The text moves to the center of the map.

To move fields to different locations in the map:

1. Move the "*DATX" field to the top line of the map (Row=1).
2. Move the "*TIMX" field to line three (Row=3), directly below the "*DATX" field.

Step 8

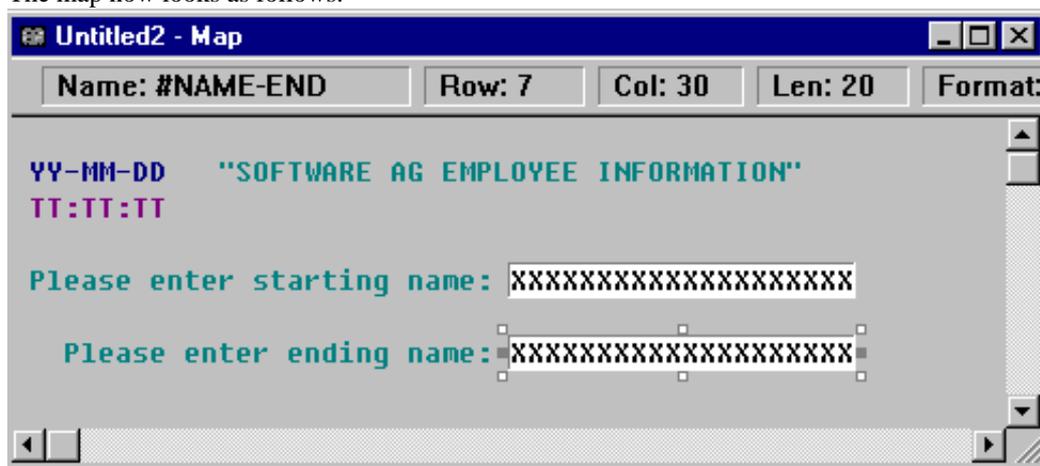
In this step, you will move ranges of fields to new locations.

▶ To position the first range of fields

1. Select a range of fields that contains the text field "PLEASE ENTER STARTING NAME:" and the "#NAME-START" data field.
Select the fields by holding down the left mouse button and dragging the mouse to surround the fields. Release the mouse button to select the fields.
2. Move the range to line five of the map (Row=5).
Move the range by placing the selector tool within the field handles and dragging the range to the new location.

▶ To position the second range of fields

1. Using the same method as above, select the text field "PLEASE ENTER ENDING NAME:" and the data field "#NAME-END".
2. Move the range to Line Seven of the map (Row=7).
The map now looks as follows.



Step 9

The first time you save a map, you must give it a name. After the map is named, you can make changes to it and save it or stow it without entering the name. If you want to save a modified map with a different name, choose "Save as" and enter a different name.

▶ To save the map and give it a name

1. From the **Object** menu, choose **Save As**.
The **Save As** dialog box appears. The current library is SYSEXP, the library where the map is saved.
2. In the "Name" text box, type "MAP01".
3. Choose **Save**.
The map is saved as MAP01 in the SYSEXP library.

Step 10

Now, you will create a processing rule for a map field.

▶ To define a processing rule for the #NAME-START data field

1. Click the #NAME-START field once to select it.
2. From the **Field** menu, choose **Rules**.
The "Field Rules" dialog box appears.
3. Choose **Create**.
A program editor window opens. 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 replaced by the name of the field.

▶ To save the processing rule and give it a rank

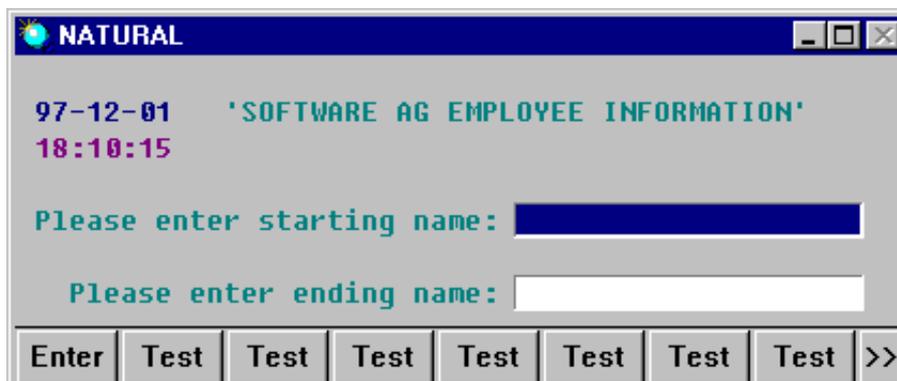
1. From the **Object** menu, choose **Save as**. The "Rule Selection" dialog box appears.
2. From the list box, select "1", and then choose **OK**.
3. Close the "Map Rule" (program editor) window by choosing **Close** from the **Object** menu.
The window closes and MAP01 reappears.

Step 11

In this step, you will test MAP01 to check whether it works as intended.

▶ To test the map

1. From the **Object** menu, choose **Test**.
The map, including the processing rule, is executed. This is the same screen that appears when the map is invoked from PGM01:



2. Type in a name and press ENTER.
You are returned to the map editor.
When you do not enter a name and press ENTER, the message "Please enter starting name" is displayed in the message line.

Step 12

When the map has been successfully tested, it has to be stowed; that is, stored in both source and object form.

To stow the map

- From the **Object** menu, choose **Stow**.

Step 13

The next step is to create a helproutine and attach it to a field in a map.

To modify the field definition for the field "#NAME-START"

1. Select the field "#NAME-START".
2. Either select **Definition** from the **Field** menu or point to the "#NAME-START" field and double-click. The "Field Definition" dialog box appears.
3. Use TAB to move to the "Help Routine:" text box, and enter "'HELP001'" (do not forget the quotation marks). "HELP001" (which is yet to be created) is the name of the helproutine that is invoked when a user presses the HELP key while the cursor is in the "#NAME-START" field.
4. Choose **OK**.
The map editor window appears.
5. Stow the map (that is, store it in source and object form) by choosing **Stow** from the **Object** menu.
6. From the **Object** menu, choose **Close**.

Step 14

Now the helproutine itself has to be created.

To create the helproutine

- Open the context menu of the "SYSEX EVT" node and select the **New > Helproutine** item.

End of Session 2.

Session 3 - Checking and Running a Program

In the previous session, you added a variable called #NAME-END to MAP01. This variable allows the program to provide an ending point for the READ statement. Otherwise, all employees from JONES to the end of the alphabet would be included in your report.

Now that the map allows both the beginning and ending name to be provided on the input screen, an IF statement must be added to the PGM01 program.

Step 1

Make sure that SYSEXPB is the current library.

In the "Programs" folder, scroll to "PGM01" and select it.

The program editor is invoked and the current version of the program PGM01 appears.

For easier editing, you can maximize the program editor window by clicking the "Maximize" button.

Step 2

The program includes the following statement:

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

Replace this statement with the following IF statement:

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

Step 3

You can add user comments to a program to identify the program modifications that you have made. A user comment helps anyone editing or maintaining a source program and is ignored during processing.

A user comment is entered by inserting a statement line or lines. If the entire line is to be reserved for a user comment, enter an asterisk and a blank (*) or two asterisks (**) in columns 1 and 2 of the line and type in the comment. If you want to place a comment in the same line of source code, separate the code from the comment with " /*" (a blank, a slash and an asterisk).

Add a comment to Line 3 to indicate that the program has been modified, for example:

```
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
```

Step 4

When you have completed the above modifications to PGM01, the program should look as follows:

```

* PGM-ID:   PGM01
* FUNCTION:  DEMONSTRATE NATURAL PROGRAM CREATION
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
* -----
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
*
  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 >= 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

```

Save the modified version of PGM01 by choosing "Save" from the "Object" menu.

Step 5

Checking a program allows you to find and correct syntax errors that would otherwise prevent the program from being compiled. In this step, you will create an error in the source code of PGM01. Then you will check the program to identify the error, correct the error, and run the program.

▶ To create an error in the PGM01 source code

1. Edit PGM01.
2. Use the arrow keys or the **Go to** function of the **Edit** menu to move the cursor to the following line:
`DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=30' #MARK`
3. Move the cursor to the second quotation mark and press **DEL** to remove the quotation mark.
Natural uses beginning and ending quotation marks to designate text strings. A text string must be closed on the same line in which it was opened. When the Natural compiler finds an odd number of quotation marks on the same line, then it reports a syntax error.
4. From the **Object** menu, choose **Check**.
When the error is detected, syntax checking is suspended. The line that contains the error is displayed, and the following error message appears:
`NAT0305 TEXT STRING MUST BEGIN AND END ON SAME LINE`

▶ To correct the error and check the program again

1. Add an quotation mark directly, and press the **CONTINUE** button.
Or press **ENTER** to return to the program editor and make the correction.
2. From the **Object** menu, choose **Check**.
When the syntax error has been corrected, and if no other syntax errors are detected, you are informed that the check was successful.
3. Choose **OK**.

Step 6

In this step, you will run the program PGM01 and view the output. When you run this program, you are prompted to enter a name. The EMPLOYEEES file is searched to locate all employees with that name; then a report that includes the Name, Department and Leave Due to each employee with that name is displayed. The names of employees who have 30 or more days leave due are marked with an asterisk.

The prompting screen is invoked at the INPUT USING MAP statement. The final report is formatted according to information in the DISPLAY statement.

The processing required to show which employees have more than 30 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.

► **To see if everything - including the map and the helproutine - works as intended**

1. From the **Object** menu, choose **Run** to compile and execute the program PGM01.
The map MAP01 is displayed.
2. Press **ENTER** without typing in anything.
The following message is displayed:
PLEASE TYPE IN A NAME
3. In the first input field in the map, enter a question mark (?).
The helproutine HELPO01 appears:
TYPE THE NAME OF AN EMPLOYEE.
4. In the first input field of the map, type the name MCKENNA, and press **ENTER**.
As there is no record with the name MCKENNA in the database, the following message is displayed:
PLEASE TRY ANOTHER NAME
5. In the first input field of the map, type the name SMITH, and press **ENTER**.
The database does include the name SMITH; the following list is displayed:

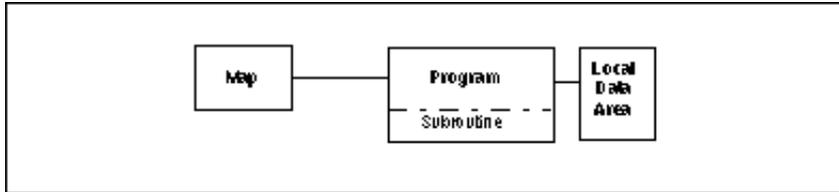
NAME	DEPARTMENT CODE	LEAVE DUE	>=30
SMITH	SALE02	28	
SMITH	FINA01	28	
SMITH	MGMT01	30	*
SMITH	TECH10	4	
SMITH	FINA01	30	*
SMITH	TECH10	8	
SMITH	TECH10	8	
SMITH	TECH10	4	
SMITH	SALE20	8	
SMITH	TECH05	8	
SMITH	MGMT10	8	
SMITH	TECH10	4	
SMITH	MGMT30	8	
SMITH	SALE20	7	
SMITH	MGMT10	8	
SMITH	SALE40	8	
SMITH	MGMT10	8	
SMITH	MGMT10	12	

MORE |

6. Press **ENTER**.
7. When the program prompts you again for a name, enter a period (.). Press **ENTER** again to return to the program editor window.
8. Close PGM01.

End of Session 3.

Session 4 - 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, however, 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, 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 later in this tutorial, 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

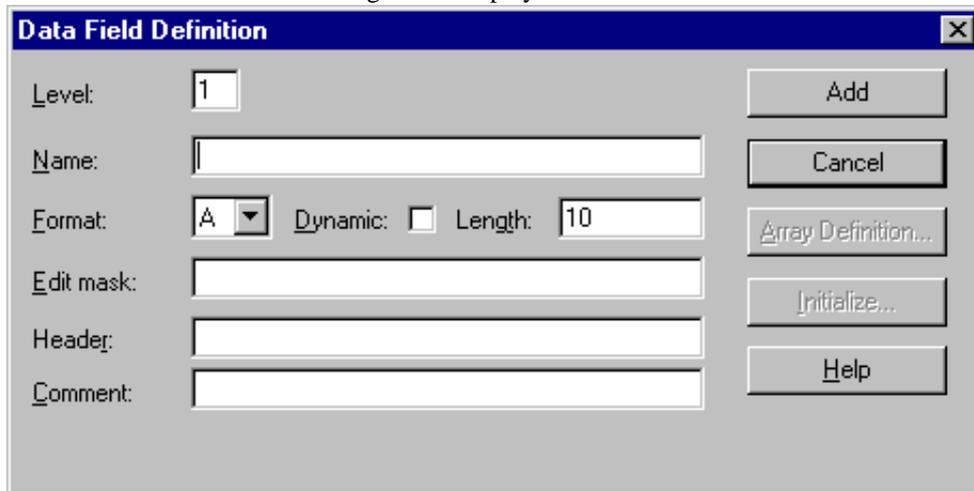
In this step, you will create a data area with three data fields. Each data field must be defined separately.

▶ To open a local data area editor window

1. From the **Object** menu, choose **New**.
2. From the cascading menu, choose **Local Data Area**.

▶ **To insert the first data field**

1. From the **Insert** menu, choose **Data Field**.
The "Data Field Definition" dialog box is displayed.



In the "Level" text box the default "1" is displayed.

2. In the "Name" text box, enter "#NAME-START".
3. Format "A" is the correct format for the "#NAME-START" data field. (Alphanumeric is the default format).
4. In the "Length" text box, enter "20".
5. Choose **Add**.

The "Define a Data Field" dialog box appears again to allow you to define another data field.

Define a second and third data field with the following attributes:

Field Name	Data Field 2	Data Field 3
Level:	1	1
Field:	#NAME-END	#MARK
Length:	20	1
Format:	A	A

When the "Data Field Definition" dialog box is displayed again, choose **Quit** to end the field definition process.

The local data area now looks as follows:

I	T	L	Name of Data Field	F	Len	Index/C
*			*** Top of Data Area ***			
1			#NAME-START	A	20	
1			#NAME-END	A	20	
1			#MARK	A	1	
*			*** End of Data Area ***			

Step 2

▶ To confirm that no syntax errors have been made

- From the **Object** menu, choose **Check**.

Step 3

Variables defined in a Natural DDM can be imported directly into the local data area.

▶ To import fields from the "EMPLOYEES" DDM

1. Select the "#MARK" field.
2. From the **Insert** menu, choose **Import**.
The "Import View" dialog box appears with the name of the current library (SYSEXPG) in the "Library" list box.
3. Open the "Library" list box and select the SYSEXDDM library.
A list of all DDMs in the SYSEXDDM library appears in the DDM list box.
4. Select the "EMPLOYEES" DDM.
A list of all the data fields in the "EMPLOYEES" DDM appears in the "Data Fields" list box.
5. Scroll through the list and select the following fields: "PERSONNEL-ID", "NAME", "DEPT", and "LEAVE-DUE".

Note:

To select individual fields, hold down **CTRL** while you click the left mouse button.

6. Choose **OK**.
The "View Definition" dialog box appears.
7. Enter "EMPLOYEES-VIEW" as the name of the view.
8. Choose **OK**.

The imported fields appear in the local data area, after the "#MARK" field. The name of the view that contains these fields (EMPLOYEES-VIEW) also appears in the data area and is identified with a V in the T (Type) column.

I	T	L	Name of Data Field	F	Len	Index
*			*** Top of Data Area ***			
	1		#NAME-START	A	20	
	1		#NAME-END	A	20	
	1		#MARK	A	1	
V	1		EMPLOYEES-VIEW			EMPLO
	2		PERSONNEL-ID	A	8	
	2		NAME	A	20	
	2		DEPT	A	6	
	2		LEAVE-DUE	N	2.0	
*			*** End of Data Area ***			

Step 4

▶ To check the new local data area

1. From the **Object** menu, choose **Check**.
2. If syntax errors are found, correct them; then check the local data area again.

Step 5

▶ To stow the new local data area

1. From the **Object** menu, choose **Stow**.
The "Stow As" dialog box appears.
2. In the "Name" text box, enter "LDA01".
As the library SYSEXP is highlighted in the "Library" list box, the LDA01 local data area will be stored in this library.
3. Choose **OK**.

Step 6

▶ To close the LDA01 local data area before continuing this session

- From the **Object** menu, choose **Close**.

Step 7

In this step, the PGM01 program is modified to reference the LDA01 local data area. After removing the lines within the DEFINE DATA statement that define variables, you will add a statement to reference the local data area.

▶ To edit PGM01

1. Open the SYSEXP library and then, from the "Objects" window, open the program PGM01.
2. Maximize the program editor window for easier editing.
3. Remove the lines that define variables:
Place the cursor at the beginning of the line containing "#NAME-START" and use the mouse to select the following text:

```

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

```

4. From the **Edit** menu, choose **Delete**.
5. Add a reference to LDA01 by entering the following statement in the blank line after LOCAL:
USING LDA01

The program should now look as follows:

```

* PGM-ID:      PGM01
* FUNCTION:    DEMONSTRATE NATURAL PROGRAM CREATION
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
* PROGRAM NOW USES A LOCAL DATA AREA
* -----
DEFINE DATA
  LOCAL
    USING LDA01
END-DEFINE
*
REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME = '.'
    ESCAPE BOTTOM
  END-IF
*
  IF #END = ' '
    MOVE #NAME TO #END
  END-IF
*
  RD1.  READ EMPLOYEES-VIEW
        BY NAME
        STARTING FROM #NAME
        THRU #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

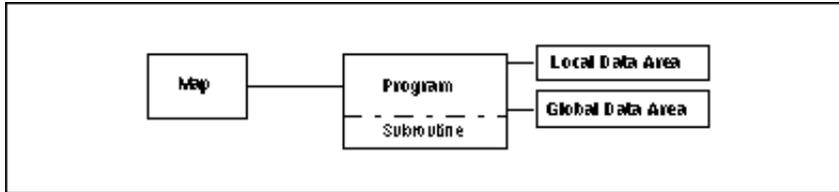
```

Step 8

1. Check the PGM01 program and correct any errors.
2. Run PGM01 to confirm that the results are the same as when the DEFINE DATA statement did not reference a local data area.
3. Stow PGM01 so that it is available for Session 5.
4. Close PGM01.

End of Session 4.

Session 5 - 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, you will modify the local data area created in the previous session. You will also modify the program so that it references not only the local data area, but also the new global data area.

Step 1

The local data area that you created in Session 4 (LDA01) is stored in the SYSEXPG library. Before you start this session, make sure that the SYSEXPG library is the current library.

You can create a new data area from an existing data area by editing the data area and saving it with a different name and type. The original data area remains unchanged, and the new data area can be edited.

In this step, you will use the local data area LDA01 to create a global data area.

Open LDA01.

▶ To save LDA01 with the name "GDA01" and change the type to "GDA"

1. From the **Object** menu, choose **Save As**.
The "Save As" dialog box appears.
2. In the "Name" text box, enter GDA01.
Do not change the name of the current library (SYSEXPG). The new global data area is stored in the SYSEXPG library.
3. Open the "Type" list box and select "Global".
4. Choose **OK**.
The data area is saved as a global data area named "GDA01". GDA01 appears in the data area editor window.

Step 2

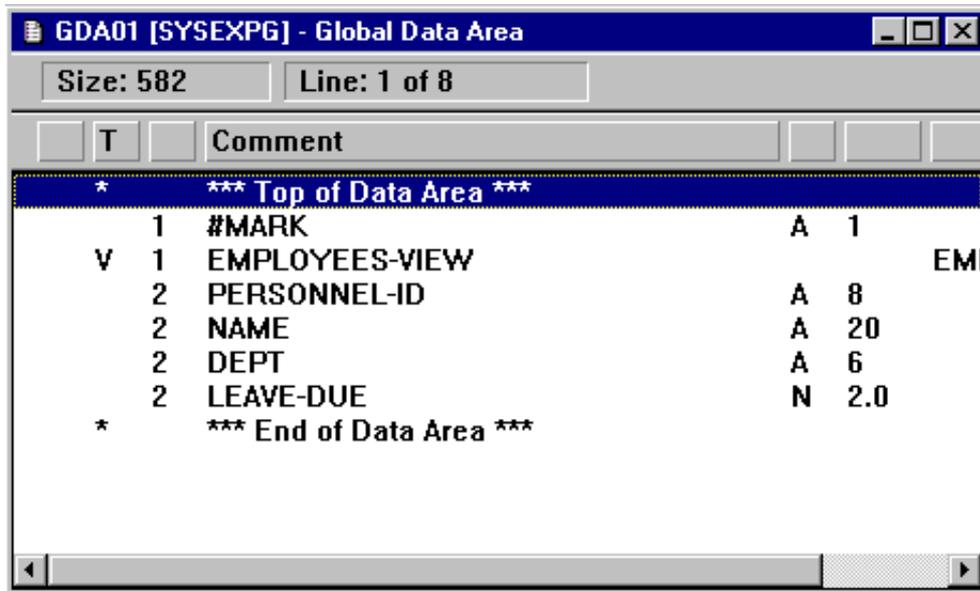
▶ To remove the data fields "#NAME-START" and "#NAME-END"

1. Select the fields "#NAME-START" and "#NAME-END".
2. From the **Edit** menu, choose **Delete**.

Note:

To select multiple fields, hold down the left mouse button and drag the mouse across the fields to be selected.

The global data area should now look as follows:



Step 3

The new data area must be stowed before any program referencing that data area can be compiled.

▶ To stow the new data area

1. Stow GDA01 by choosing "Stow" from the "Object" menu.
2. Close GDA01 by choosing "Close" from the "Object" menu.

Step 4

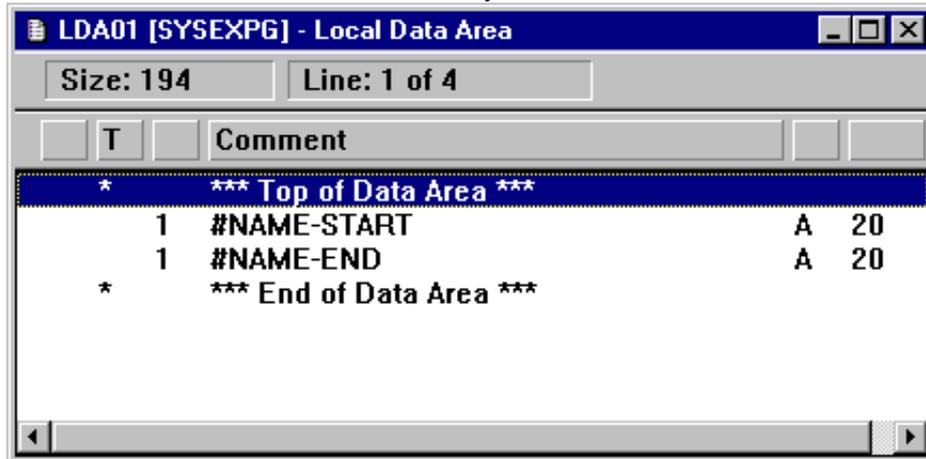
Now that the new global data area has been created, the variables contained in it must be removed from the local data area.

Open LDA01.

▶ **To remove all the data fields that are now in the global data area GDA01 ("#MARK", "EMPLOYEES-VIEW", and all remaining lines)**

1. Select all fields except "#NAME-START" and "#NAME-END".
2. From the **Edit** menu, choose **Delete**.

The revised local data area now contains only the variables "#NAME-START" and "#NAME-END":



3. Store the revised local data area.
LDA01 is now ready to be referenced by the program PGM01.
4. Close LDA01.

Step 5

The DEFINE DATA statement in the PGM01 program must now reference data that are located in the global data area GDA01 as well as the local data area LDA01.

▶ **To open PGM01, and add a reference to the global data area**

1. Open PGM01.
2. Place the cursor at the end of the DEFINE DATA statement and press ENTER.
3. In the blank line created, type GLOBAL USING GDA01 and press ENTER.

Step 6

In this step, you will revise the output instructions in PGM01.

In this step, you will modify the program PGM01 to include a WRITE TITLE statement, which produces a multiple-line title in the resulting report, and modify the format of the DISPLAY statement.

To do so

1. Insert a blank line after the following lines:

```
RESET #MARK  
END-IF
```

2. Add the following WRITE TITLE statement:

```
WRITE TITLE  
  / '*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***'  
  / '*** ARE MARKED WITH AN ASTERISK ***' //
```

The "/" notation indicates a line break. The title lines are centered and are not underlined.

3. Change the DISPLAY statement as follows:

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

The revised program should now have the changes to the DEFINE DATA, WRITE TITLE, DISPLAY statements, and the program header (comment) as shown below.

```

* PGM-ID:    PGM01
* FUNCTION:   DEMONSTRATE NATURAL PROGRAM CREATION
* A BEGINNING AND ENDING NAME ARE USED FOR THE OUTPUT
* 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 = '.'
    ESCAPE BOTTOM
  END-IF
*
  IF #END = ' '
    MOVE #NAME TO #END
  END-IF
*
  RD1.  READ EMPLOYEES-VIEW
        BY NAME
        STARTING FROM #NAME
        THRU #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

After you have completed all changes:

1. Check the program and correct any errors that might exist.
2. Run the program, using "SMITH" as the name on the input screen.
Note the differences in the report output, which should have the following format:

The screenshot shows a window titled "NATURAL" with a report output. The report title is "*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***" and a subtitle is "*** ARE MARKED WITH AN ASTERISK ***". The report is a table with columns for NAME, DEPT, and LU DUE. Asterisks are placed to the right of the LU DUE values for SMITH in departments MGMT01 and FINA01.

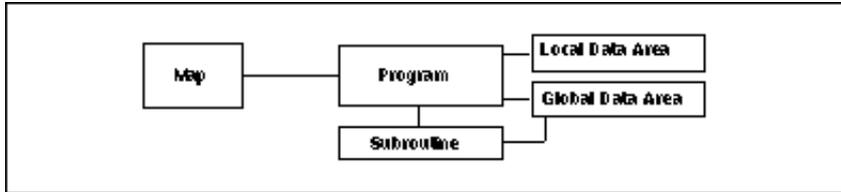
NAME	DEPT	LU DUE	*
SMITH	SALE02	28	
SMITH	FINA01	28	
SMITH	MGMT01	30	*
SMITH	TECH10	4	
SMITH	FINA01	30	*
SMITH	TECH10	8	
SMITH	TECH10	8	
SMITH	TECH10	4	
SMITH	SALE20	8	
SMITH	TECH05	8	
SMITH	MGMT10	8	
SMITH	TECH10	4	
SMITH	MGMT30	8	
SMITH	SALE20	7	

MORE |

3. After you have confirmed that PGM01 has no errors, stow it for future modification in Session 6 and close PGM01.

End of Session 5.

Session 6 - 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

In this step, you will create a subroutine named SUBR01:

Note:

This subprogram is contained in library SYSEXP. If you have access to this library, you do not have to perform this step.

▶ To open a new program editor window

1. From the **Object** menu, choose **New**.
2. From the cascading menu, choose **Subroutine**.
3. Enter the following statements:


```

* SUBR-ID:  SUBR01
*
* FUNCTION:  DEMONSTRATE NATURAL
* THIS IS A SUBROUTINE
*
*
*
* -----
      
```

▶ To save the subroutine

1. From the **Object** menu, choose **Save As**.
The "Save As" dialog box appears.
2. In the "Name" text box, enter "SUBR01".
SUBR01 should be saved in the SYSEXP library. If SYSEXP is not the current library, from the "Library" list box, select SYSEXP.
3. Choose **OK**.

Step 2

In this step, you will edit the program PGM01, copy two statements and paste them into the subroutine SUBR01.

▶ To edit the program PGM01

1. Use the **Minimize** button to minimize SUBR01.

Note:

You can reopen SUBR01 by clicking its icon or by choosing "SUBR01" from the "Window" menu.

2. Open PGM01.

▶ To copy the DEFINE DATA statement

1. Place the cursor at the beginning of the DEFINE DATA statement and drag the mouse until the following lines are selected:

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

2. From the **Edit** menu, choose **Copy**.
The DEFINE DATA statement is copied and placed on the clipboard.
3. Use the **Minimize** button to minimize PGM01.

▶ To paste the copied statement into SUBR01

1. From the **Window** menu, choose **SUBR01**.
2. Place the cursor below the last comment line.
3. From the **Edit** menu, choose **Paste**.
The DEFINE DATA statement appears.
4. Cut the following DEFINE SUBROUTINE block from PGM01 and paste it into SUBR01. Follow the same procedure as above but, from the **Edit** menu, choose **Cut** instead of **Copy**.

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

5. Paste the block below the END-DEFINE in program PGM01.
6. Add an END statement at the end of the subroutine.

Step 3

The subroutine SUBR01 should now appear as follows:

```
* SUBR-ID:  SUBR01
*
* FUNCTION: DEMONSTRATE NATURAL
* THIS IS A SUBROUTINE
*
*
*
* -----
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
*
END
```

1. From the **Object** menu, choose **Check** to check SUBR01 and correct any errors.
2. From the **Object** menu, choose **Stow** to stow SUBR01.
3. From the **Object** menu, choose **Close** to close SUBR01.

The program PGM01 should now look as follows:

```

* PGM-ID:   PGM01
* FUNCTION: DEMONSTRATE NATURAL
* 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
*
  IF #NAME-END = ' '
    MOVE #NAME TO #NAME-END
  END-IF
*
  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

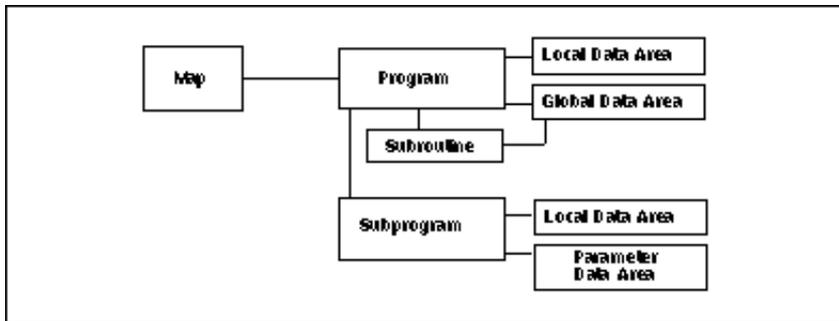
```

Step 4

1. Check PGM01 and correct any errors.
2. Run the program to confirm that the results are the same with an external subroutine as with an internal subroutine.
3. Stow the program for the next session.
4. Close PGM01, saving your changes.

End of Session 6.

Session 7 - Invoking a Subprogram



In Natural, both subprograms and subroutines can be invoked from a main program.

A subprogram is invoked using a CALLNAT statement. Data are passed from the main program (the calling program) to a subprogram through a set of parameters that are referenced or defined in the DEFINE DATA PARAMETER statement of the subprogram.

While a subroutine such as SUBR01 created in Session 6 shares a global data area with the main program, the subprogram only receives data that are passed by way of a parameter list from the main program's CALLNAT statement.

In this session, the PGM01 program will be expanded to include a CALLNAT statement that invokes a subprogram. In the subprogram, the employees identified from the main program will be the basis of a FIND request to the VEHICLES file. As a result, your report will contain VEHICLES information from the subprogram as well as leave due, etc. from the main program.

The new subprogram will require the creation of a local data area and a parameter data area. In this case, new variables will be defined in the main program's local data area, and this will in turn help create the subprogram's parameter data area variables.

Step 1

The local data area that you created in Session 4 (LDA01) is stored in the SYSEXP library. Make sure that the SYSEXP library is the current library.

In this step, you will modify the LDA01 local data area to accommodate the new subprogram. The following fields must be added to LDA01:

```
#PERS-ID
#MAKE
#MODEL
```

These fields are referenced in the CALLNAT statement that you will add to the program PGM01 in a later step.

Open LDA01.

▶ **To add the data fields**

1. Select the field "#NAME-END".
2. From the **Insert** menu, choose **Data Field**.
The "Data Field Definition" dialog box appears.
In the "Level" text box, the default "1" is displayed.
3. In the "Name" text box, enter "#PERS-ID".
4. In the "Length" text box, enter "8".
5. Choose **Add**.
The field definition you entered is added to the LDA01 local data area, and the "Data Field Definition" dialog box reappears, allowing you to define the next data field.

▶ **To define the two remaining fields, "#MAKE" and "#MODEL", as you defined the "#PERS-ID" field, enter a length of "20" for each field**

1. Choose **Quit** to close the "Data Field Definition" dialog box and return to the data area editor window.
The local data area should now appear as follows.

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 ***

2. Check and store the LDA01 local data area.

Step 2

With minor modifications, the LDA01 local data area can be used to create the parameter data area that will be needed for the subprogram.

In this step, you will delete two of the data fields in LDA01, then save the revised data area as a parameter data area named PDA01. The original LDA01 local data area remains intact. (It is also possible to define the parameter data area directly by using the menu to choose "Object > New > Parameter data area").

Open LDA01.

▶ **To delete the data fields "#NAME-START" and "#NAME-END"**

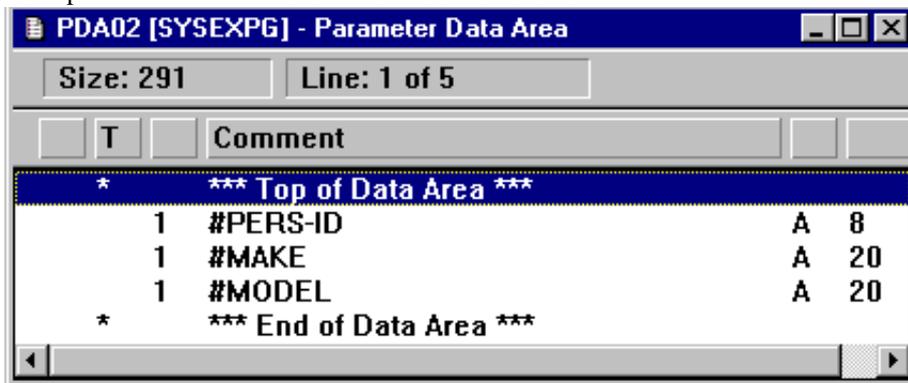
1. Select the fields "#NAME-START" and "#NAME-END".
2. From the **Edit** menu, choose **Delete**.

▶ **To save the data area with the name PDA02 and data area type "Parameter"**

1. From the **Object** menu, choose **Save As**.
The "Save As" dialog box appears.
2. In the "Name" text box, enter "PDA02".
3. Open the "Type" list box and select "Parameter".

4. Choose **OK**.

Your parameter data area should now look as follows.



5. Check the new parameter data area and correct any errors.
6. In the SYSEXP library, stow the parameter data area.
7. Close the parameter data area.

Step 3

The subprogram will also use variables that are local to the program. In this step, you will create a new local data area.

▶ To open a new data area window to create a local data area

1. From the **Object** menu, choose **New**.
2. From the cascading menu, choose **Local data area**.

Step 4

Fields contained in any Natural DDM can be imported into a data area. In this step, you will import several fields from the VEHICLES DDM into the new local data area.

To import fields from the VEHICLES DDM

1. From the **Insert** menu, choose **Import**.
The "Import View" dialog box appears with the name of the current library (SYSEXPB) in the "Library" list box.
2. Open the "Library" list box and select the SYSEXDDM library.
A list of all DDMs in the SYSEXDDM library appears in the DDM list box.
3. Select the "VEHICLES" DDM.
A list of all the data fields in the "VEHICLES" DDM appears in the "Data fields" list box.
4. Select the fields "PERSONNEL-ID" through "MODEL" (drag the mouse across the fields to select them) and choose OK.
5. In the "View Definition" dialog, choose **OK**.
The fields appear in the data area window.

The local data area now contains fields imported from the "VEHICLES" DDM as shown below:

LDA02 [SYSEXP] - Local Data Area				
Size: 485		Line: 1 of 7		
T	Comment			
*	*** Top of Data Area ***			
V	1	VEHICLES		VEHICLES
	2	PERSONNEL-ID	A 8	
G	2	CAR-DETAILS		
	3	MAKE	A 20	
	3	MODEL	A 20	
*	*** End of Data Area ***			

Step 5

To save the new local data area as LDA02

1. From the **Object** menu, choose **Save As**.
The "Save As" dialog box appears.
2. In the "Name" text box, enter "LDA02".
3. Choose **OK**.
The local data area is saved as LDA02 in the SYSEXPB library.
4. Check the new local data area and correct any errors.
5. Stow the new local data area.
LDA02 is now ready for use by the subprogram.
6. Close LDA02.

Step 6

The subprogram used in this session, SPGM02, receives the personnel number passed by the main program (PGM01) and uses this number as the basis for a search of the VEHICLES file.

The SYSEXPB demo library should include the SPGM02 subprogram.

If SPGM02 is available, ensure that it has been stowed and then proceed directly to Step 7 (modifying the main program) later in this session.

If SPGM02 is not available, you can create it. Instructions are provided below.

▶ **To open a new program editor window to create the subprogram**

1. From the **Object** menu, choose **New**.
2. From the cascading menu, choose **Subprogram**.
3. Enter the subprogram shown below:

```
* PGM-ID:  SPGM02
* -----
DEFINE DATA
  PARAMETER
    USING PDA02
  LOCAL
    USING LDA02
END-DEFINE
*
FD1.  FIND (1) VEHICLES
      WITH PERSONNEL-ID = #PERS-ID
      MOVE MAKE (FD1.)   TO  #MAKE
      MOVE MODEL (FD1.) TO  #MODEL
      ESCAPE BOTTOM
END-FIND
*
END
```

4. Save SPGM02 and stow it.
5. Close SPGM02.

Step 7

In this step, you will modify the main program (PGM01) to accommodate the subprogram.

▶ **To do so**

1. Open PGM01.
2. Add the following statements immediately before the **WRITE TITLE** statement:


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

The parameters passed in the **CALLNAT** statement come from both the global data area and the local data area. Also, the variables defined in the parameter data area of the subprogram do not have to have the same name as the variables in the **CALLNAT** statement. Because the parameters are passed by address, it is only necessary that they match in sequence, format, and length.

Because the subprogram is now returning vehicle information, the DISPLAY statement must be modified as shown below:

```
*
WRITE TITLE
  / '*** PERSONS WITH 30 OR MORE DAYS LEAVE DUE ***'
  / '*** ARE MARKED WITH AN ASTERISK ***' //
DISPLAY 1X '//N A M E' NAME
        1X '//DEPT' DEPT
        1X '//LV/DUE' LEAVE-DUE
        ' ' #MARK
        1X '//MAKE' #MAKE
        1X '//MODEL' #MODEL
```

1. Check PGM01 and correct any errors.
2. Run PGM01.
3. Stow PGM01. Close PGM01.

End of Session 7.