A Natural Tutorial

By Jochen Stein
August 2009

What is Natural?

Natural is a programming language designed to simplify the implementation of business solutions. It takes a very pragmatic and non-theoretical approach to common programming tasks such as database access.

Natural is similar to JAVA in that it is an interpreted language; during execution Natural source programs are dynamically "compiled" into a platform independent byte code (object) format which is then executed in a platform dependent environment.

While originally developed for mainframe computers, Natural has been available on Windows and UNIX platforms for several years.

Install the software

This tutorial uses Natural on Windows and although it can access many different types of databases (e.g. DB2 on mainframe computers and RDBMS databases using SQL) Natural is optimised to access Adabas databases. In this tutorial we will use an Adabas database.

You can download free Community Editions of both Natural and Adabas from the download section of the Adabas and Natural Developer Community site.

Install Natural

Download the package, unzip it and choose "setup.exe" in the folder where you unpacked it to.

Once installation begins you will be asked to choose which Natural products you wish to install. Simply choose "Natural". Please refer to Figure 1.

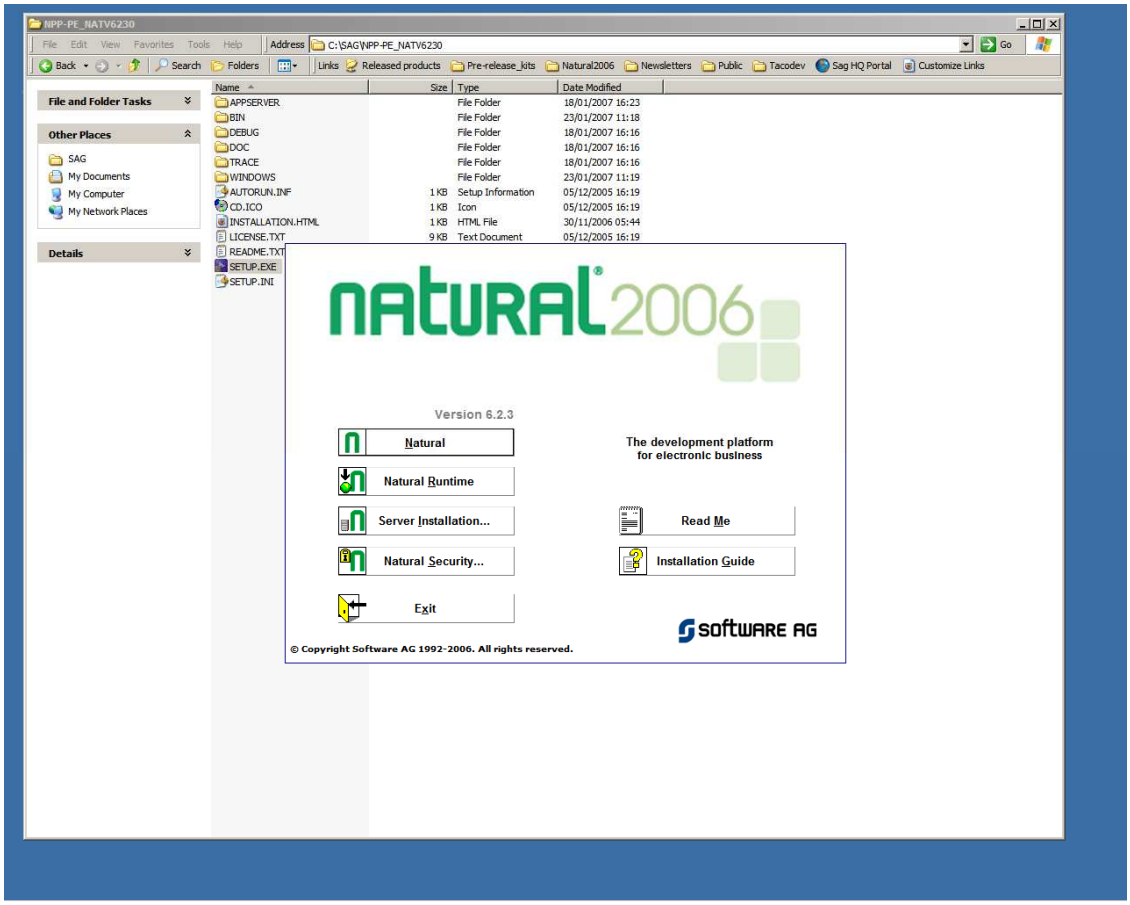

Figure 1 - Natural installer

Next, you will see a couple of questions regarding the Terms of Agreement. You may accept or reject them as you wish; however, if you reject them the installation will terminate!!

Then you will be asked to supply a license key. Use the license file you were given when you downloaded the Natural Community Edition (this is an xml file). Then choose "Standard" when asked what type of Natural installation you want.

After this, simply accept all the defaults that the installation provides.

The Natural installation placed some icons on your desktop. One is labelled Natural (the Natural Developer Studio), plus one for the Natural Configuration Utility and one for the Natural Component Browser.

Configure Natural

Before we start the Natural Studio we need to do some configuration to make life a little easier for us.

Natural distinguishes between two programming modes – Reporting and Structured mode. All of our examples will use structured mode. In order to enable this permanently we need to start the Natural Configuration Utility by clicking on its icon.

Open the trees "Natural Parameter Files" and "NATPARM" by clicking on the "+" signs in front of the text. Then click on the "+" signs in front of "Natural Development Environment" and "Compiler Options". Next, click the box to check "Structured Mode" on the menu on the right hand side of the window. Please refer to Figure 2.

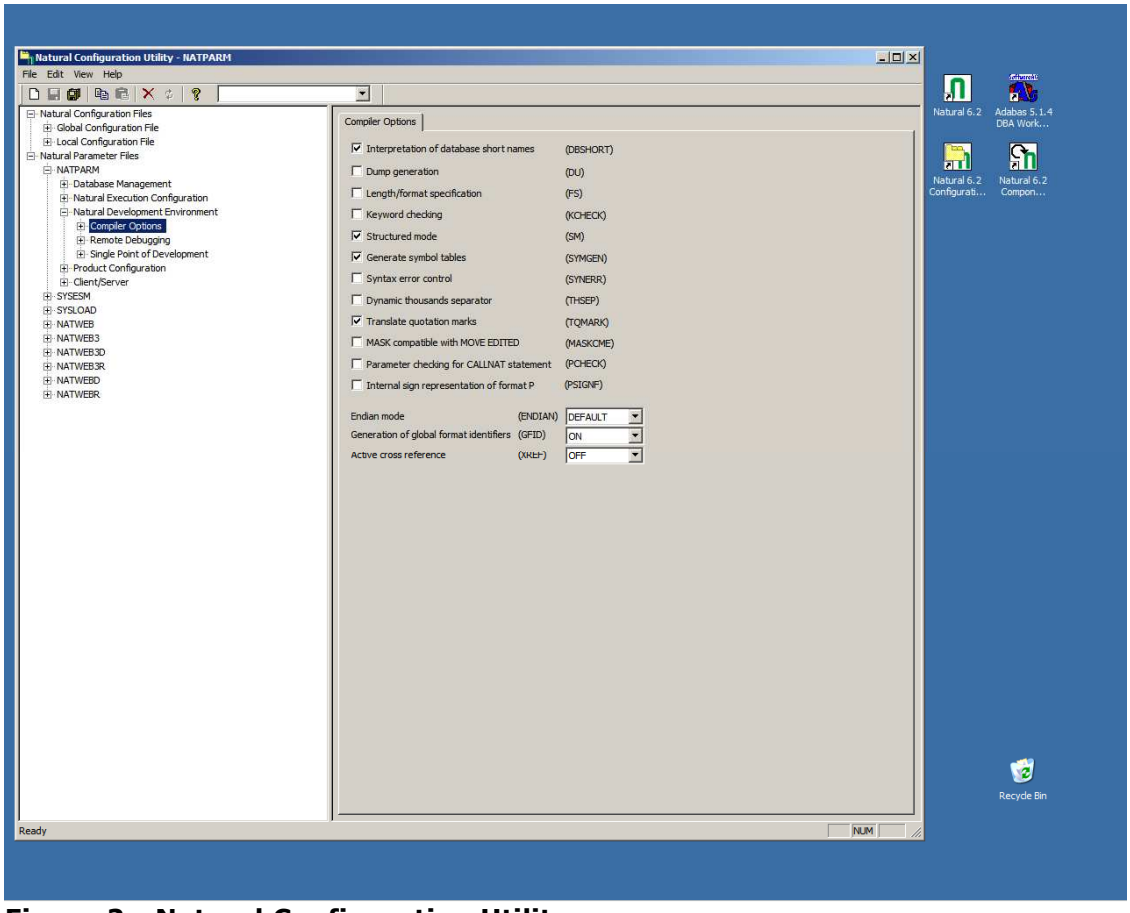

Figure 2 - Natural Configuration Utility

While you are here you can also check the "Generate symbol tables" box. This is not absolutely necessary but it makes debugging a little easier. See Figure 2.

Go to the menu options along the top, click on "File", save these settings and exit the Configuration Utility.

Test the Natural installation

Now we can start the Natural Developer Studio. You do this by simply clicking on the Natural icon on your desktop. You should get something which looks like this Figure 3:

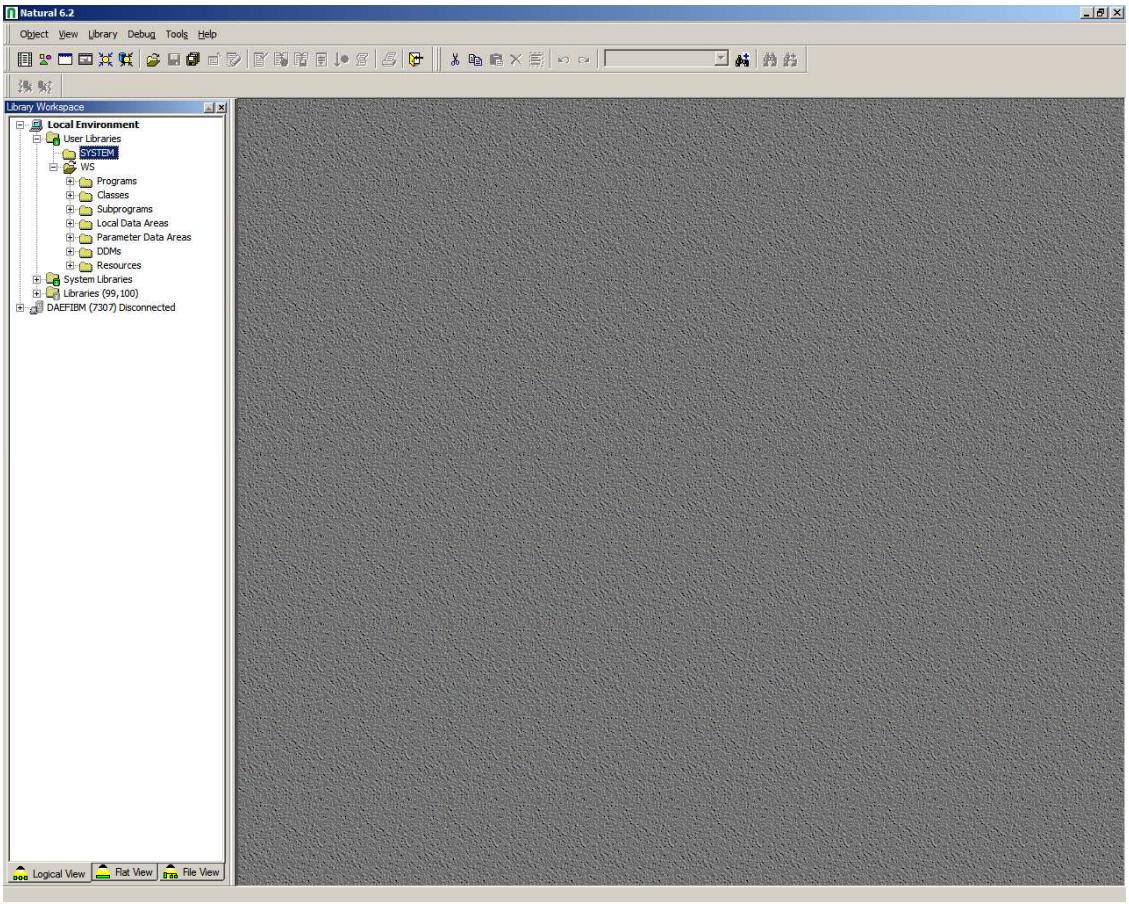

Figure 3 - Typical Natural Studio screen

Now there is only one thing left for us to do. I find it is sometimes useful to have the Natural command line available. You can view this by selecting "View" on the top menu bar and then checking the "Command Line" option.

You have now configured Natural. So let's get on with Adabas….

Install Adabas

After downloading and unzipping the package, again simply choose "setup.exe". After this, simply accept all the defaults that the installation provides.

The Adabas installation placed an icon for the *Adabas DBA workbench* under *Start->All Programs->Software AG Adabas v6.1.n* on your machine. You may also want to place a short cut to this on your desktop because we will need it later.

Start the sample Adabas database

Adabas delivers a Sample Adabas database, which has to be installed after Adabas. To install this sample database, you go to *Start->All Programs->Software AG Adabas v6.1.n* on your machine and click the icon "Create Demo Database". The Sample Database will now create in a Command Window.

We will now use the DBA workbench to start this database, so simply click on the DBA workbench icon you placed on your desktop. You can now start the database by selecting the database and then "Database" from the top menu bar and then "Start", Figure 4.

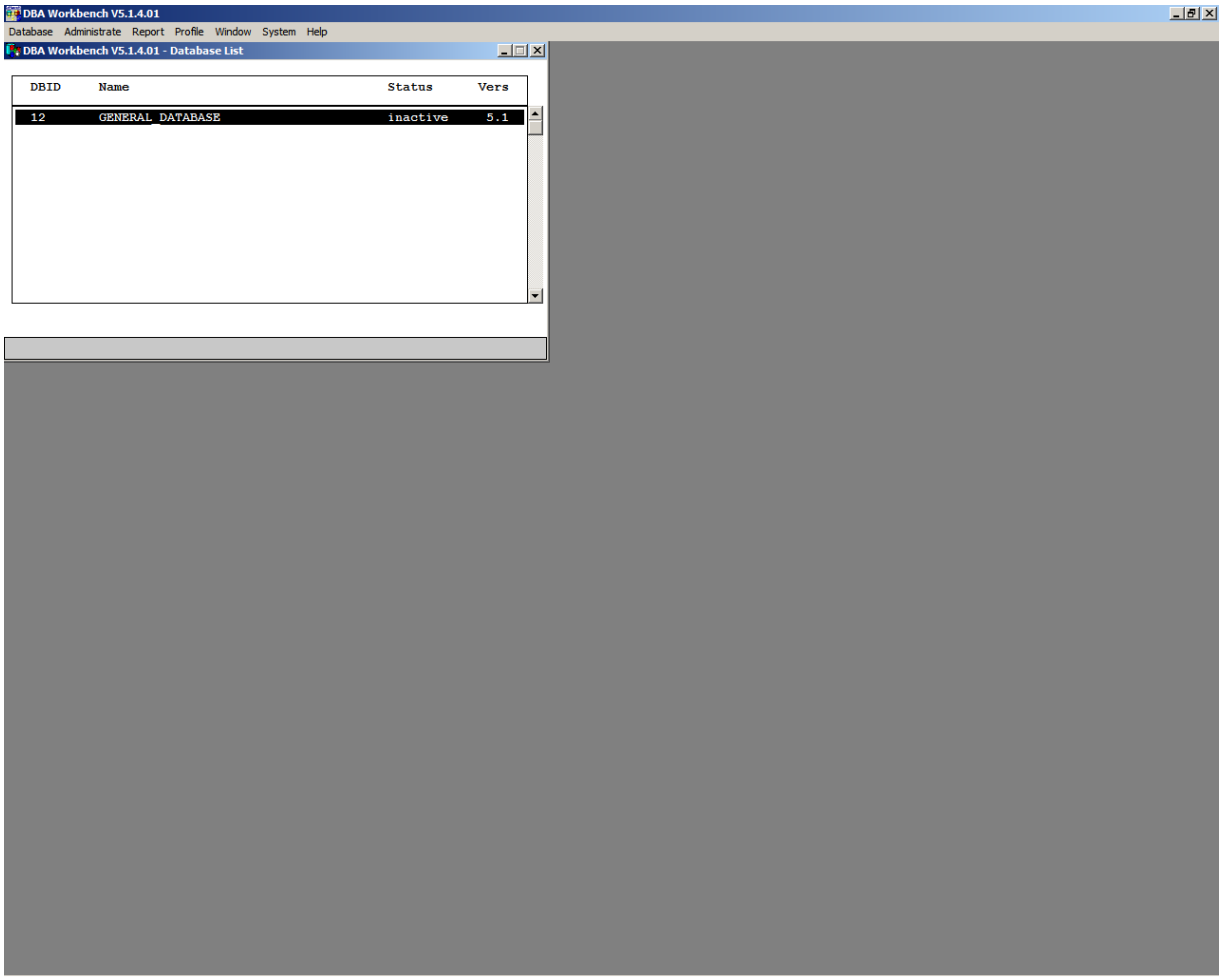

Figure 4 - Using the DBA workbench to start an Adabas database

When you want to terminate the database simply select "Database" and "Stop". You will then be asked how you want to terminate the database. Choose option "Shutdown".

Congratulations! You have now successfully installed Natural and Adabas!!

Take a break, you've earned it, and then we'll get onto writing a simple Natural program.

Natural library structure

Before we go ahead and create a Natural program, please bear with me while I tell you a little about the structure of a Natural application.

Natural uses libraries as containers to store objects (programs, classes, subprograms, etc.). These libraries are managed in the panel on the left hand side of the Natural Studio screen, called the library workspace. Please refer to Figure 5.

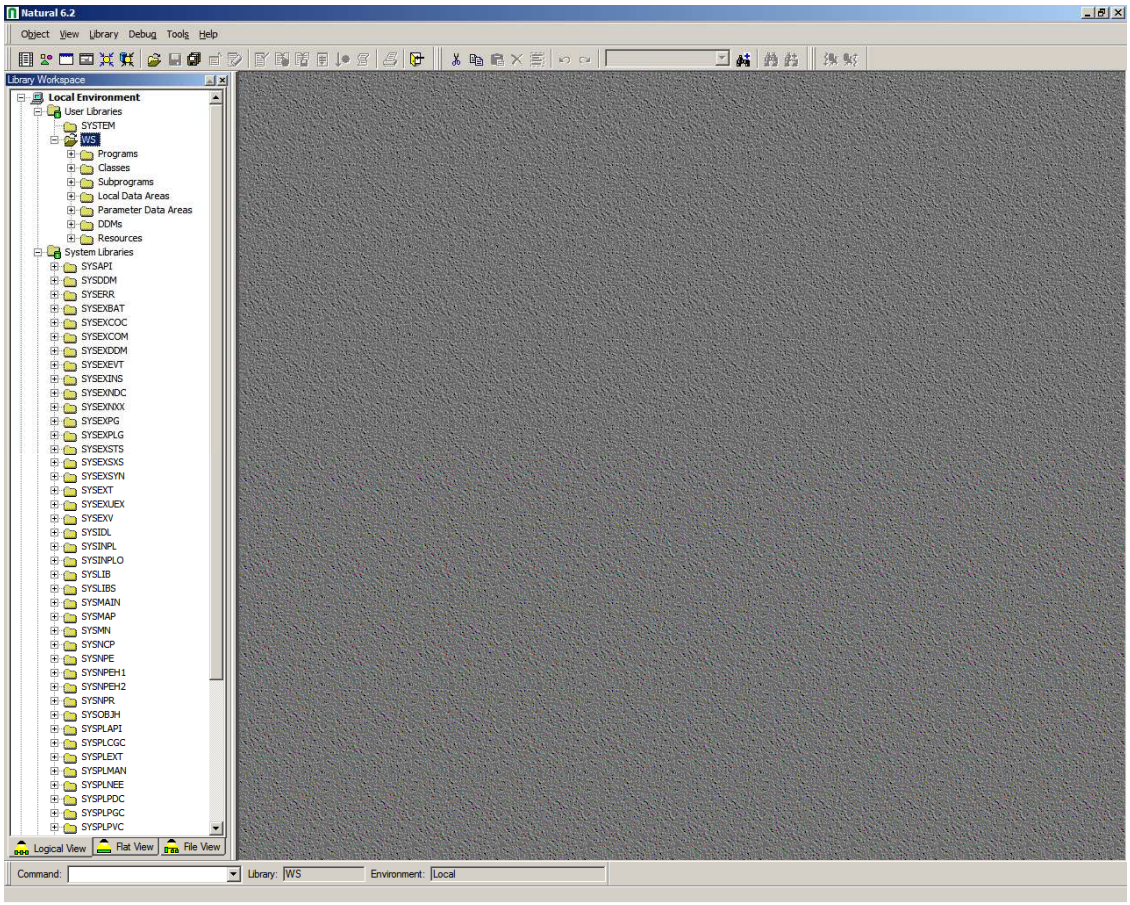

Figure 5 - Typical Natural Studio Screen showing the Library Workspace panel

You can see that there are different types of libraries shown in the workspace but the ones you need to know about are:

System Libraries: These contain the Natural objects required to run Natural tools and utilities.

System libraries always start with "SYS" and are reserved for Software AG. Please do not modify their contents and do not use them to store your own objects.

User Libraries: A user library contains all the objects which make up a user application. Typical object types are programs, classes, subprograms, data areas, etc. We will see later how you create and use these items.

User libraries are the libraries we will be concerned with in this tutorial, so now let's look at how to set up your own user library.

Select the node named "User Libraries" in the library workspace view and then from the “Library” menu, choose “New”, Figure 6.

Alternatively, you can invoke the context menu (right mouse click) and choose “New” from the drop down menu.

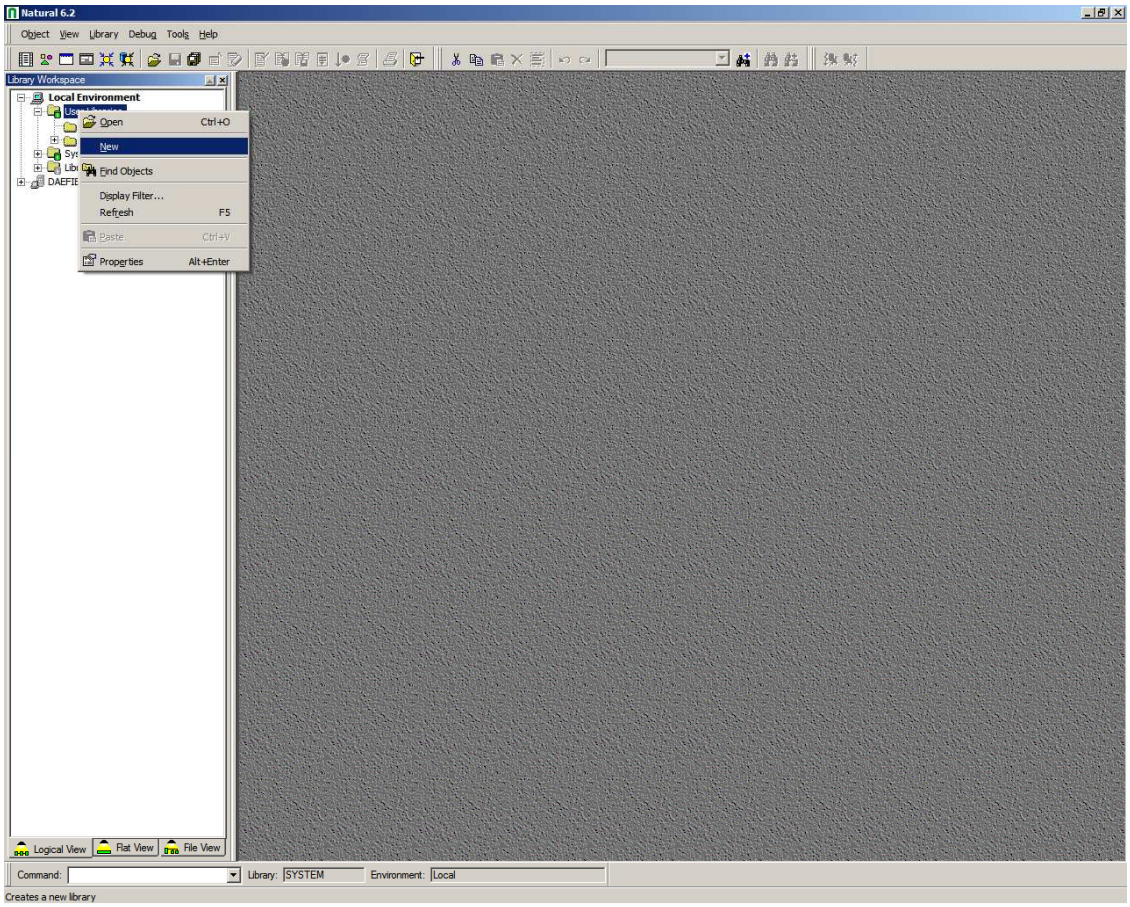

Figure 6 - Creating a new Natural user library

A new library with the default name "USRNEW" is created and is shown in the User Libraries tree. The default name is selected so that you can immediately enter a new name by simply overwriting the default.

Let’s call our library “TUTORIAL”.

The Natural version of “Hello World”

Everybody’s first attempt at programming in a new language is the “Hello World” program and here we are going to be no exception.

You first need to create a new program and open up an editor window so we can input some Natural statements.

Do this by selecting the “TUTORIAL” library you just created in the library workspace panel and then either choose “New” and “Program” from the context menu, or select the “New program” icon in the program toolbar, or select “Object” in the main toolbar and then choose “New” and “Program”.

Whichever option you choose will start up the program editor and present you with an empty program editor window like this, Figure 7.

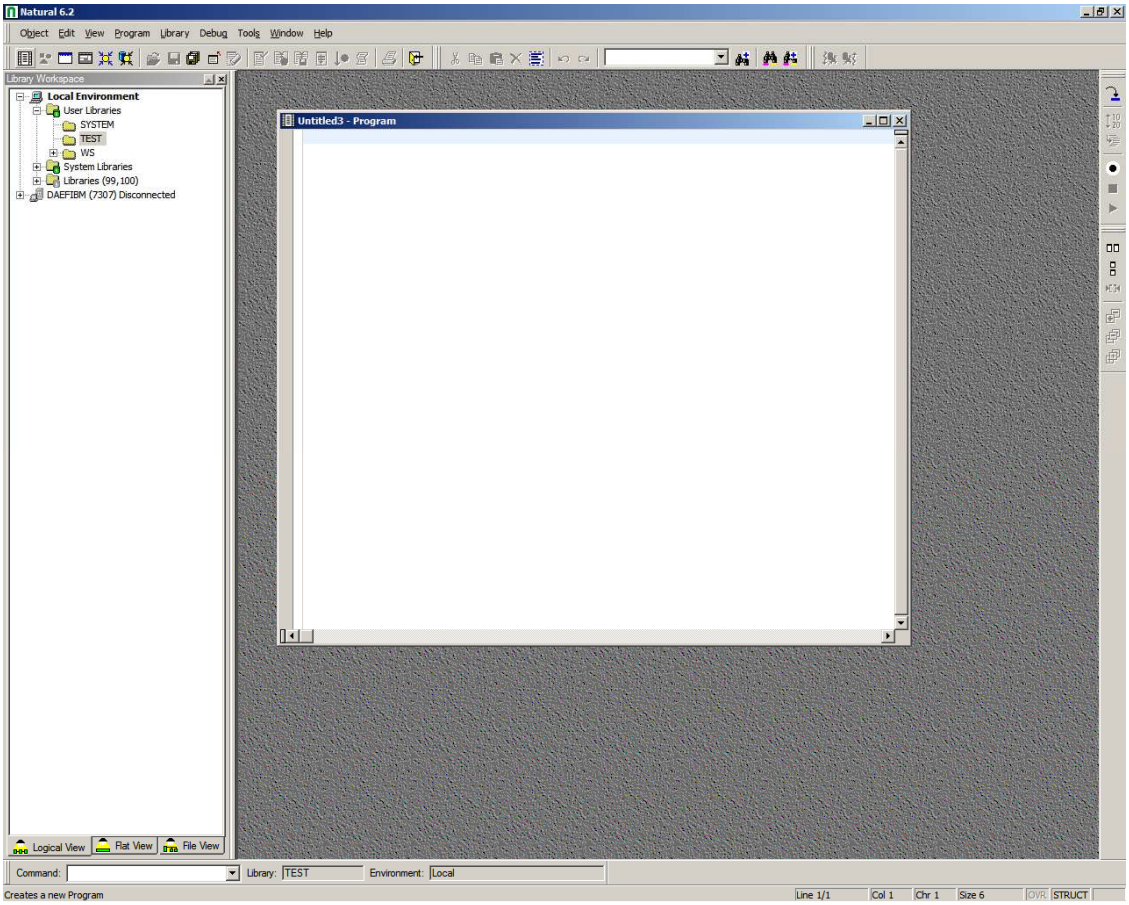

Figure 7 - An empty program editor screen

Now you can input your Natural statements, so enter the following code:

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

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

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

The text that is to be shown in the output is defined with the WRITE statement. It is enclosed in quotation marks. The END statement is used to mark the physical end of a Natural program. Each program must end with END.

Now we want to see if we have made any typing errors and, if there aren't any, run our program. For this we can use the system command RUN. This automatically first invokes the system command CHECK which checks the program code for errors and, if no error is found, the program is then compiled on the fly and executed. See Figure 8.

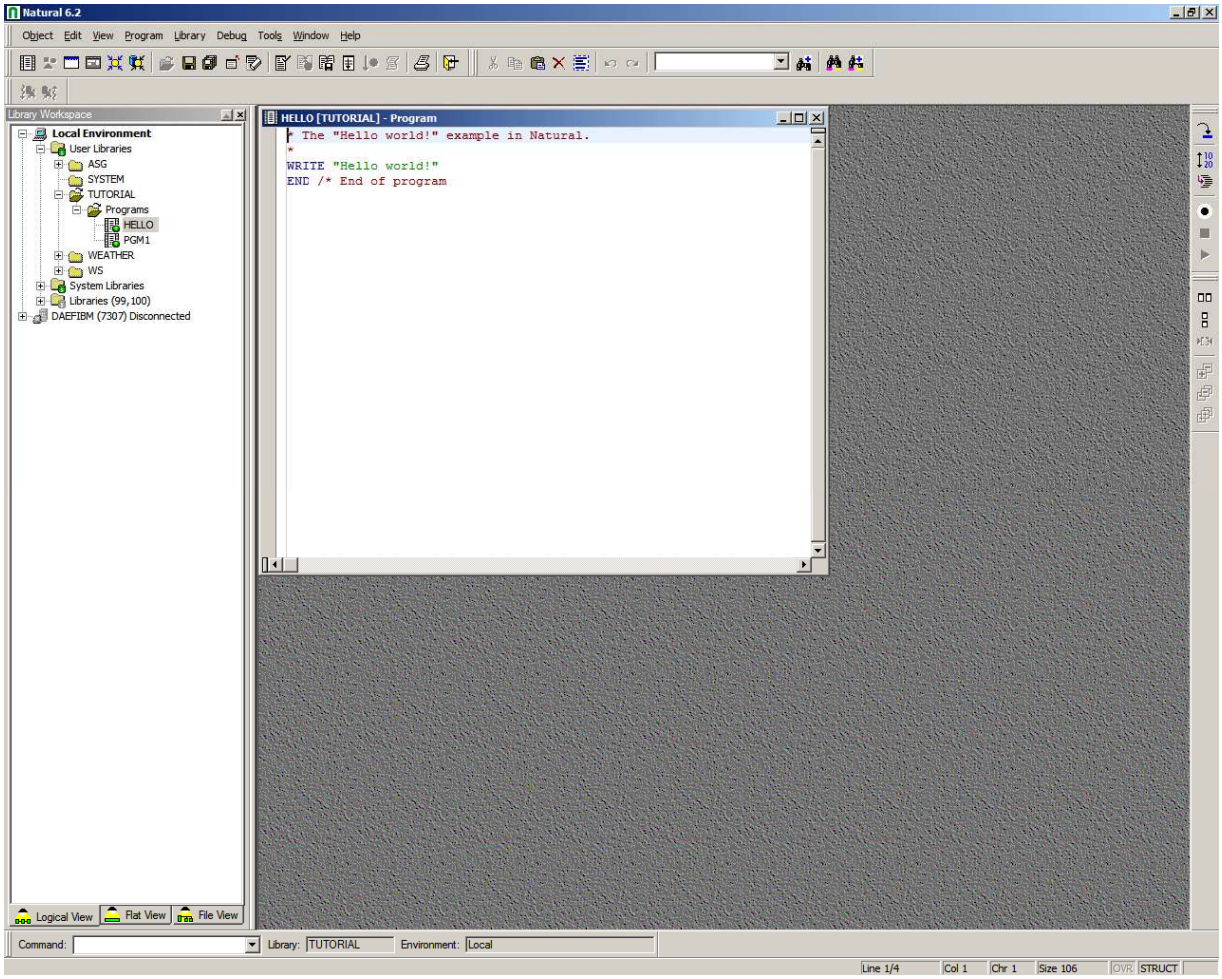

Figure 8 - Running a Natural program

You invoke the RUN command by selecting "Object" and "Run" or by simply choosing the RUN icon on the program toolbar (Figure 8).

The Natural program is started and the output is displayed in a terminal emulation screen, Figure 9.

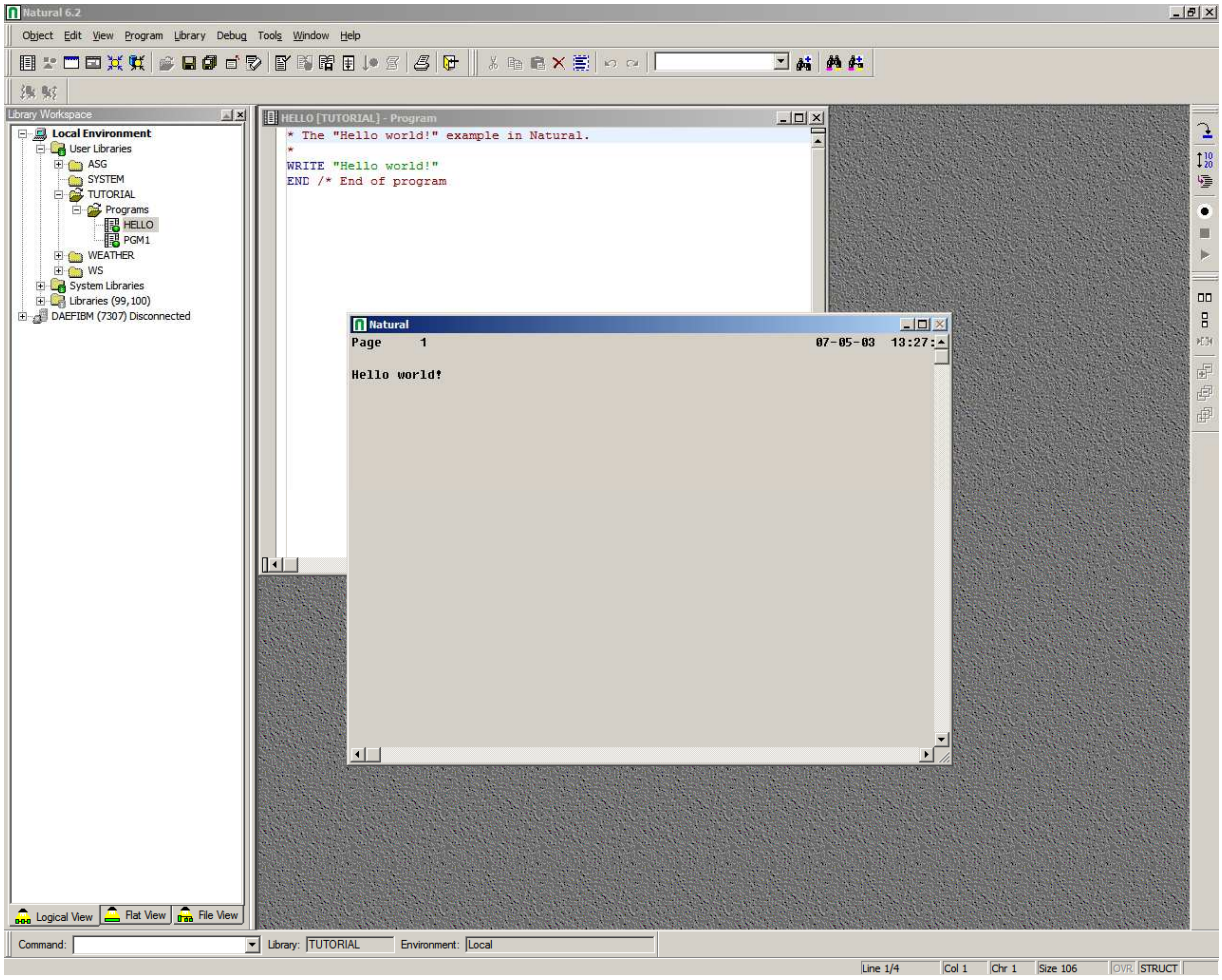

Figure 9 - "Hello World" program output

Now press "Enter" to return to the program editor.

Finally we want to save our program. We have seen that Natural is an interpretive language, so you can run the source you typed in directly without having to go through a compile stage as you would with, for example Cobol or C.

So we can now simply save the program by choosing "Object" and "Save As".

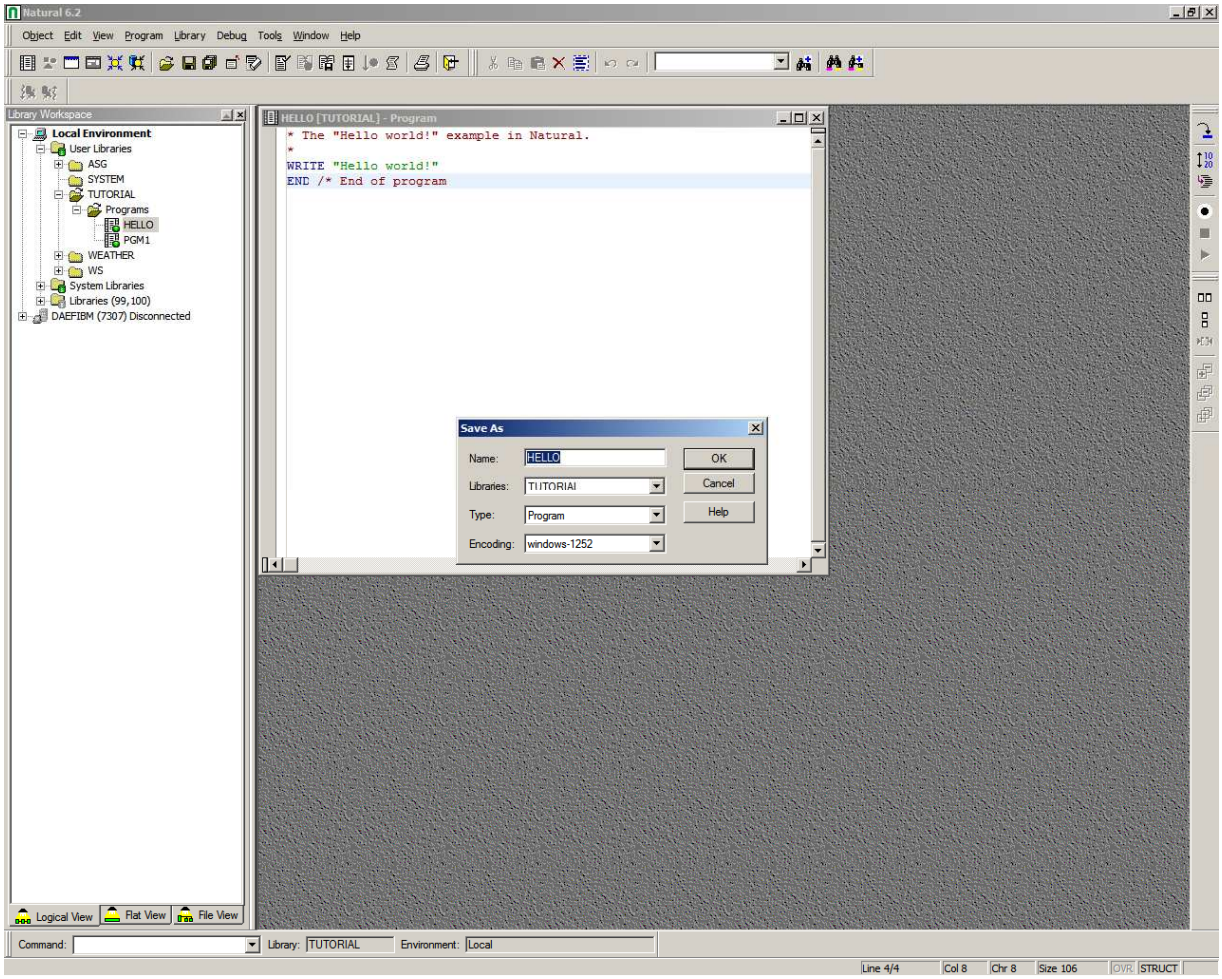

Figure 10 - Saving the Hello World program

The Save As command pops up a window to ask us for a name for our program, let's be inventive and call it "HELLO". See Figure 10.

You will now see an icon labelled “HELLO” representing your program under the user library “TUTORIAL” in the left hand Library Workspace window, Figure 11 (click on TUTORIAL, then Programs).

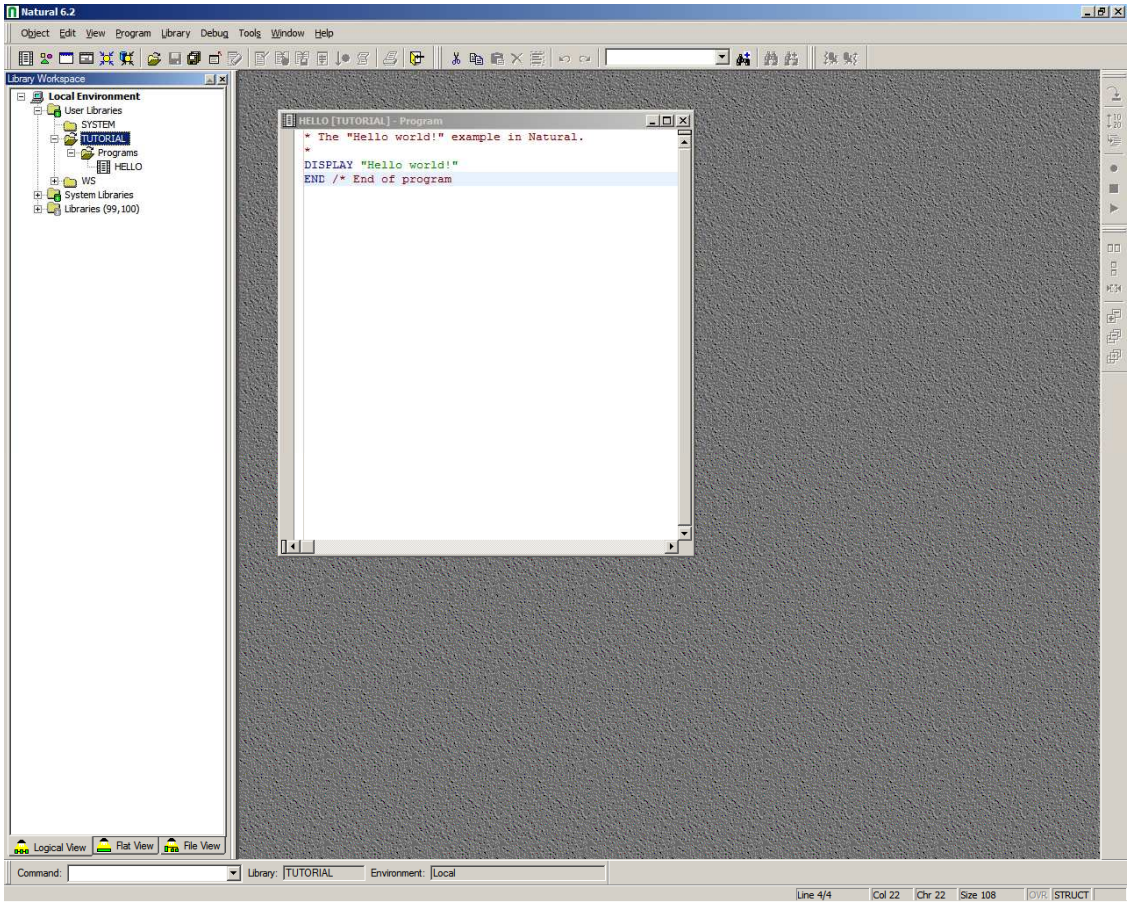

Figure 11 - Result of saving the HELLO program

You can now close the program editor window.

A bit more about Natural programs

In order to re-run your program you have to open the source file and use the run command as described above.

There is, however, another option in Natural and that is to “compile” the program source into an intermediate object form. In Natural this is called cataloging. You can then invoke this intermediate object directly without having to open the program source.

So now, let’s catalog your program by selecting the Program icon folder under TUTORIAL in the Library Workspace and then either choose the Catalog icon on the program toolbar or select Catalog from the list of command options (under Library on main menu bar).

Notice now that a small green dot has appeared in the program icon to indicate that the catalog function has been performed on this item.

You can now use the execute command, again either from the program menu or from the drop down command option list, to run your program. The results should be exactly as before.

This two stage process of saving and then cataloguing a program can be a bit tedious so Natural also provides a function to do everything in one step. This is the STOW command; this saves our program but also invokes the CHECK command before saving the program and the CAT command afterwards. The STOW icon is located on the program toolbar and the command is also offered on the drop down menus. From now on we will use this more convenient method of saving our programs.

That's it. You have now successfully created and run your first Natural program. That wasn't too difficult was it?

Using Natural to access an Adabas database

Before we go onto creating a program which uses the database we created I want to say just a short few words about how Natural interacts with the world of databases and Adabas in particular.

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

To be able to use the database fields in a Natural program, you must specify the fields from the DDM in a view.

DDMs are usually defined by the Natural administrator but some sample DDMs are already provided for us in the system library SYSEXDDM. For this tutorial we will use the predefined DDM for the EMPLOYEES database file. This also conveniently happens to be one of the files we loaded into our test database.

Start by creating a new program.

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

So enter the following code in the program editor:

```
DEFINE DATA
LOCAL

END-DEFINE
*
END
```

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

Now you can import the fields, including the required format and length definitions, from the DDM into the program editor. To do this you simply place the cursor in the line below LOCAL and from the Program menu, choose "Import". See Figure 12.

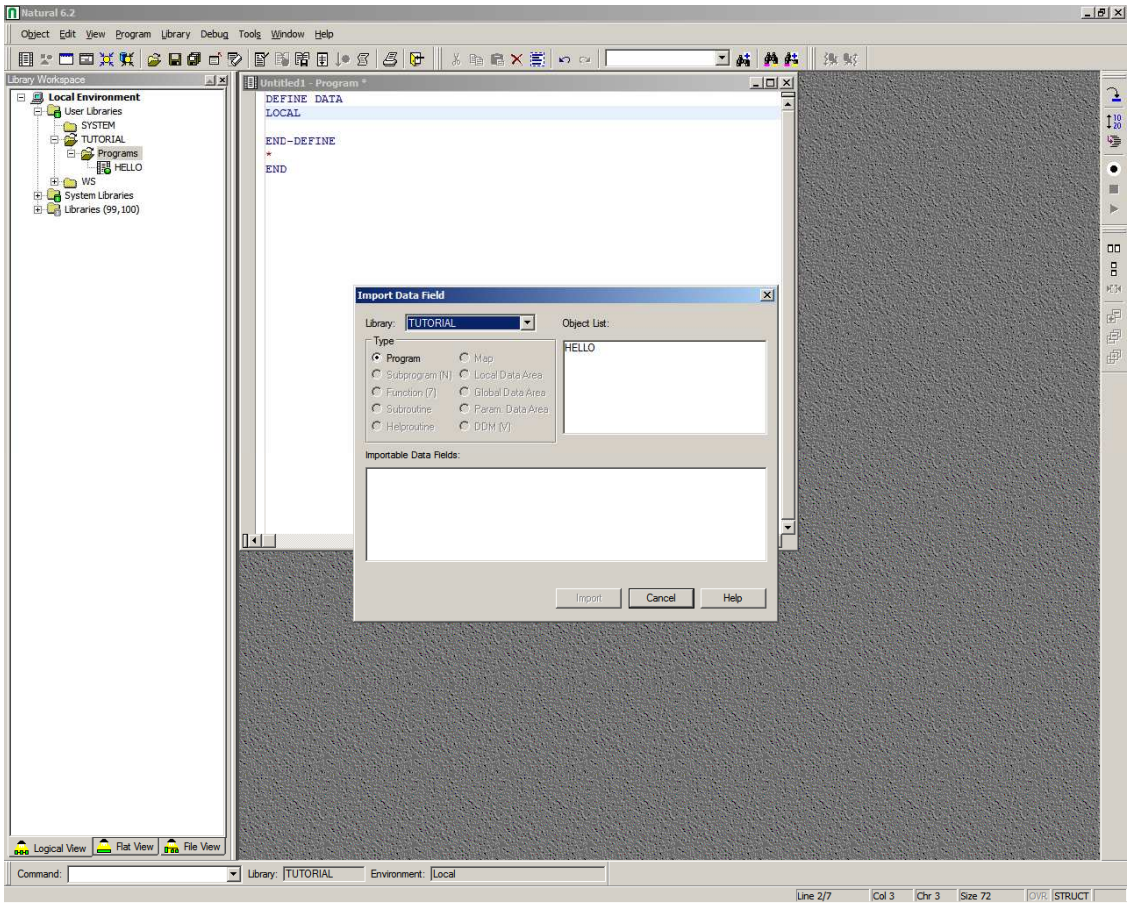

Figure 12 - How to import a DDM into a Natural program

The Library field is automatically filled to the current library, but we know the DDM we want to use is in SYSEXDDM. So in the drop-down list box, select SYSEXDDM.

Now select the DDM option button and all defined DDMs are shown in the Object list box.

Select the sample DDM with the name "EMPLOYEES". See Figure 13.

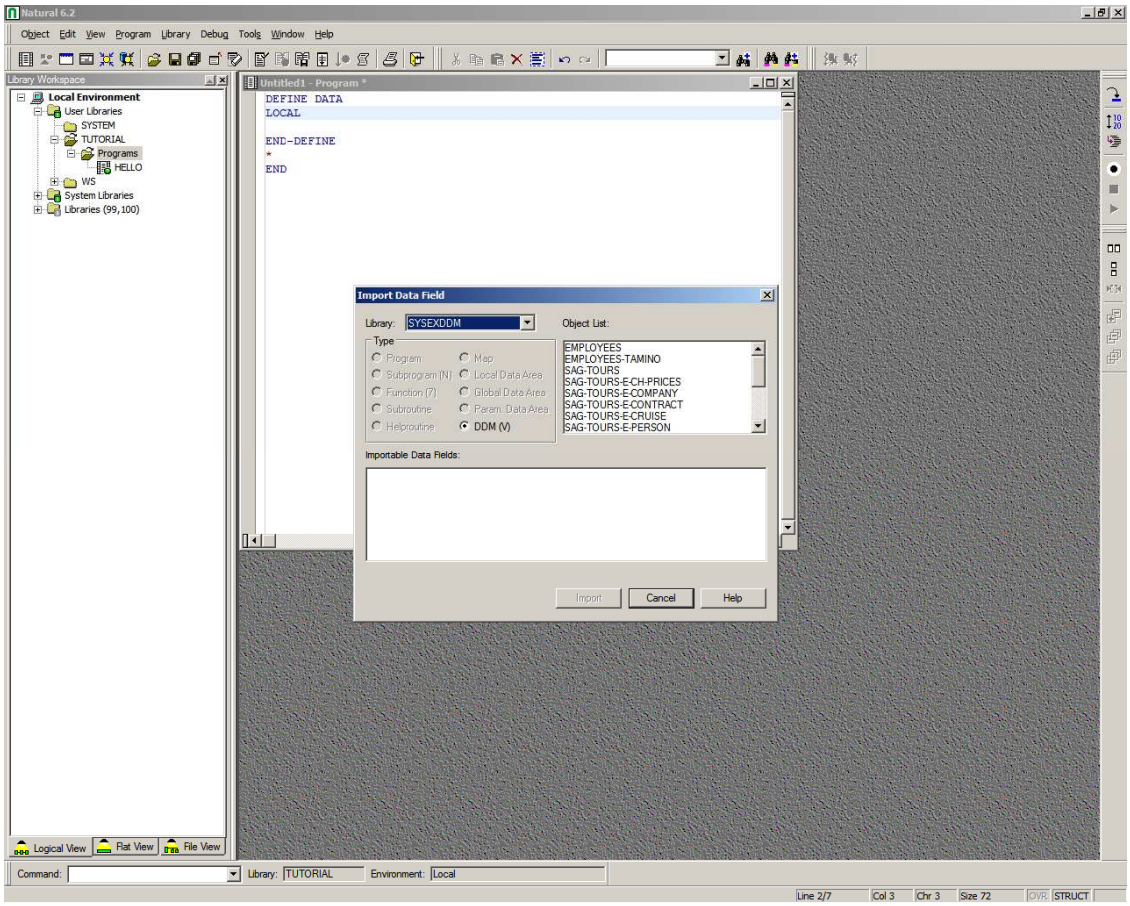

Figure 13 – How to import a DDM into a Natural program

The importable data fields are now shown at the bottom of the dialog box, Figure 14.

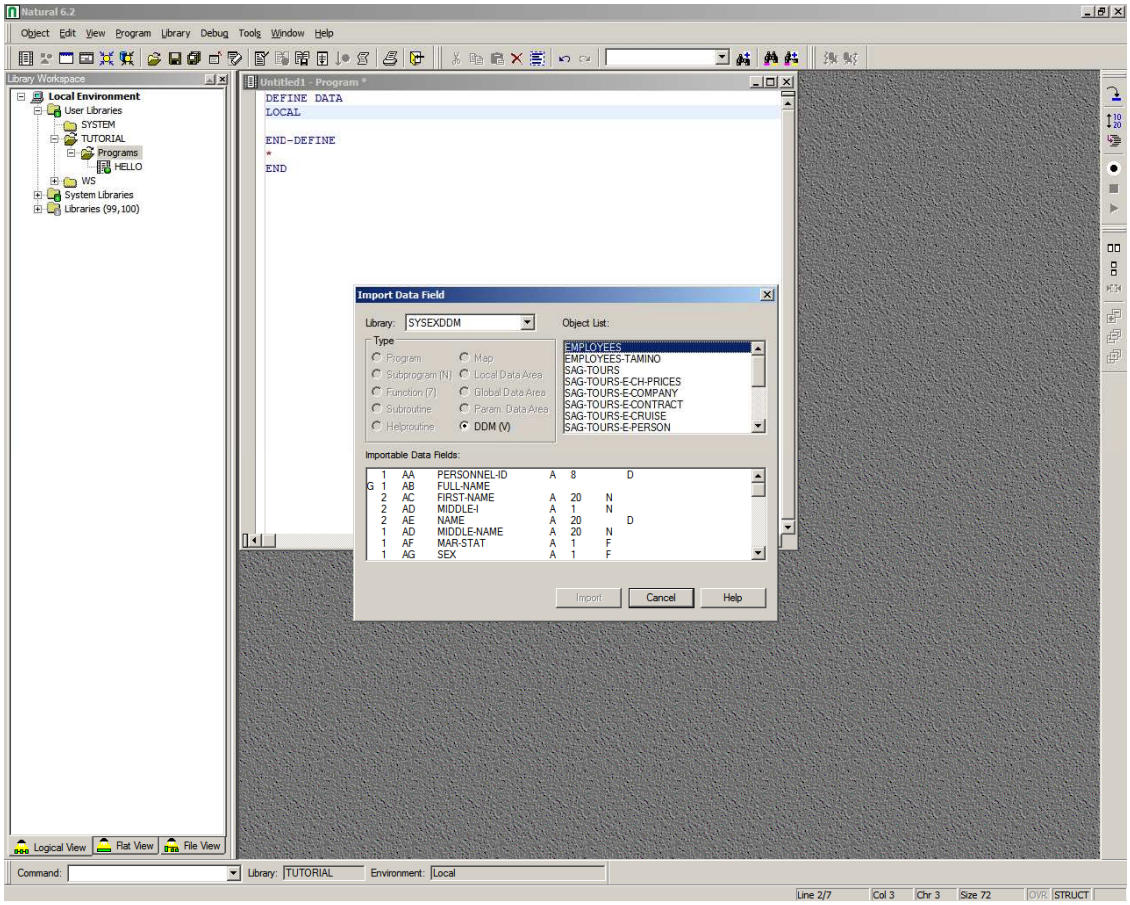

Figure 14 - How to select the DDM data fields

Press CTRL and select the following fields:

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

Choose the Import button.

The View Definition dialog box appears and by default the name of the DDM is proposed as the view name. You can specify any other name; we want to call our view "EMPLOYEES-VIEW".

Choose the OK button.

The Cancel button in the Import Data Field dialog box has now been changed to Quit. So choose the Quit button to close the Import Data Field dialog box.

The following code should have been inserted in the program editor:

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

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

Levels are used in conjunction with field grouping. Fields assigned a level number of 2 or greater are considered to be a part of the immediately preceding group which has been assigned a lower level number.

You can also see a little on data definitions in Natural, the format and length of each field is indicated in parentheses, A stands for alphanumeric, and N stands for numeric.

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

So insert the following code below END-DEFINE:

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

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

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

Your program should now look something like this:

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

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

*
END
```

This is all you need. Before you run your program, make sure the Adabas database we created has been started (it needs to be active) and you started Natural in Structure Mode. Now, if you run your program the following output will appear, Figure 15:

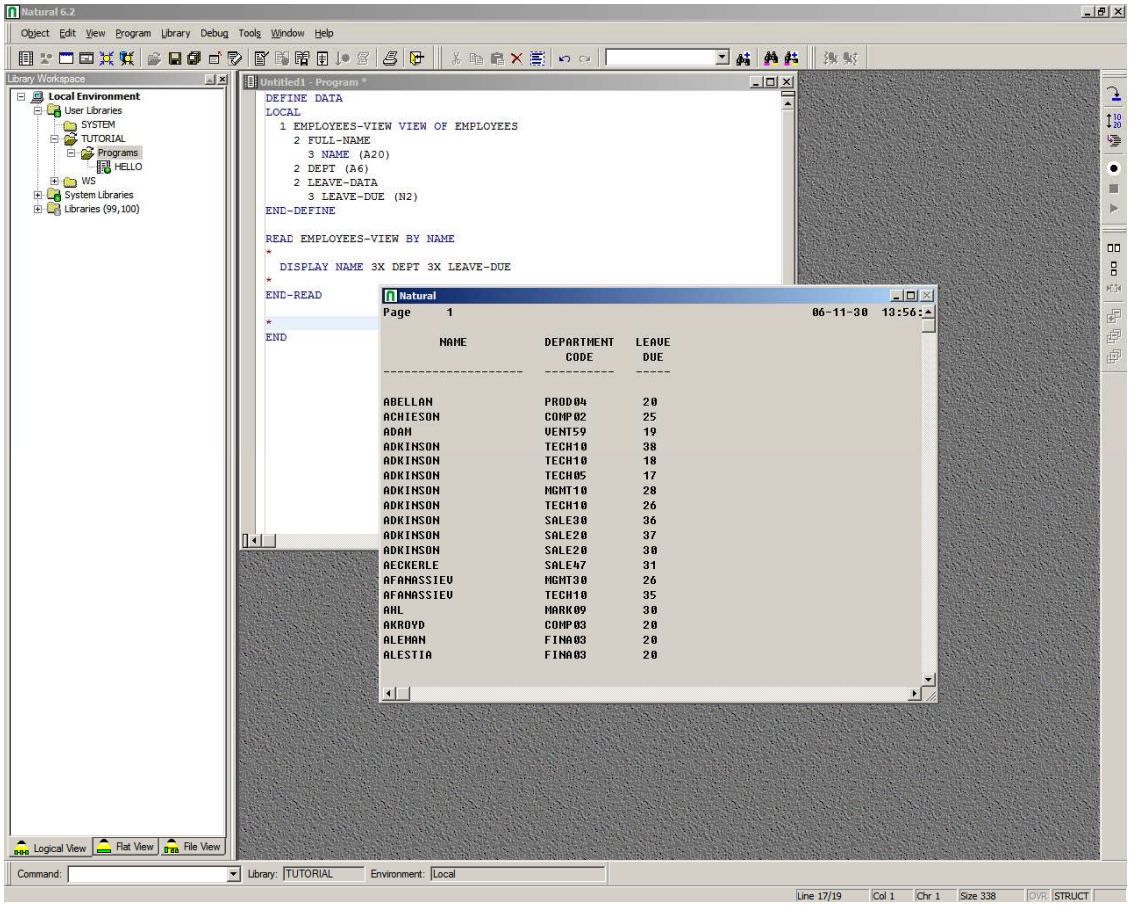

Figure 15 - Sample output from the database

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

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

Press ENTER repeatedly to display all pages, or press Esc to return to the program editor.

Now save this program with the STOW command; let's call it PGM1.

You will have noticed that the previous output was very long and it would be a good idea if we could restrict it so that only the data for a range of values is displayed.

So we will now only display names with values starting with "Adkinson" and ending with "Bennett". I happen to know that these names are defined in the demo database.

Unlike, for example PHP or Ruby, Natural is a strictly typed language and performs type checking to ensure that variables are used consistently. So before you can use new variables, you have to define them. Therefore, insert the following below LOCAL:

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

These are user-defined variables; they are not defined in demo database.

Note that we use the hash (#) at the beginning of the name just to distinguish the user-defined variables from the fields defined in the demo database. It is not required.

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

Now insert the following below the READ statement:

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

Your program should now look as follows:

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

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

READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ

*
END
```

Now run this and scroll through the output by hitting Enter. You will notice that the Program displays output up to the name "Bennett", Figure 16.

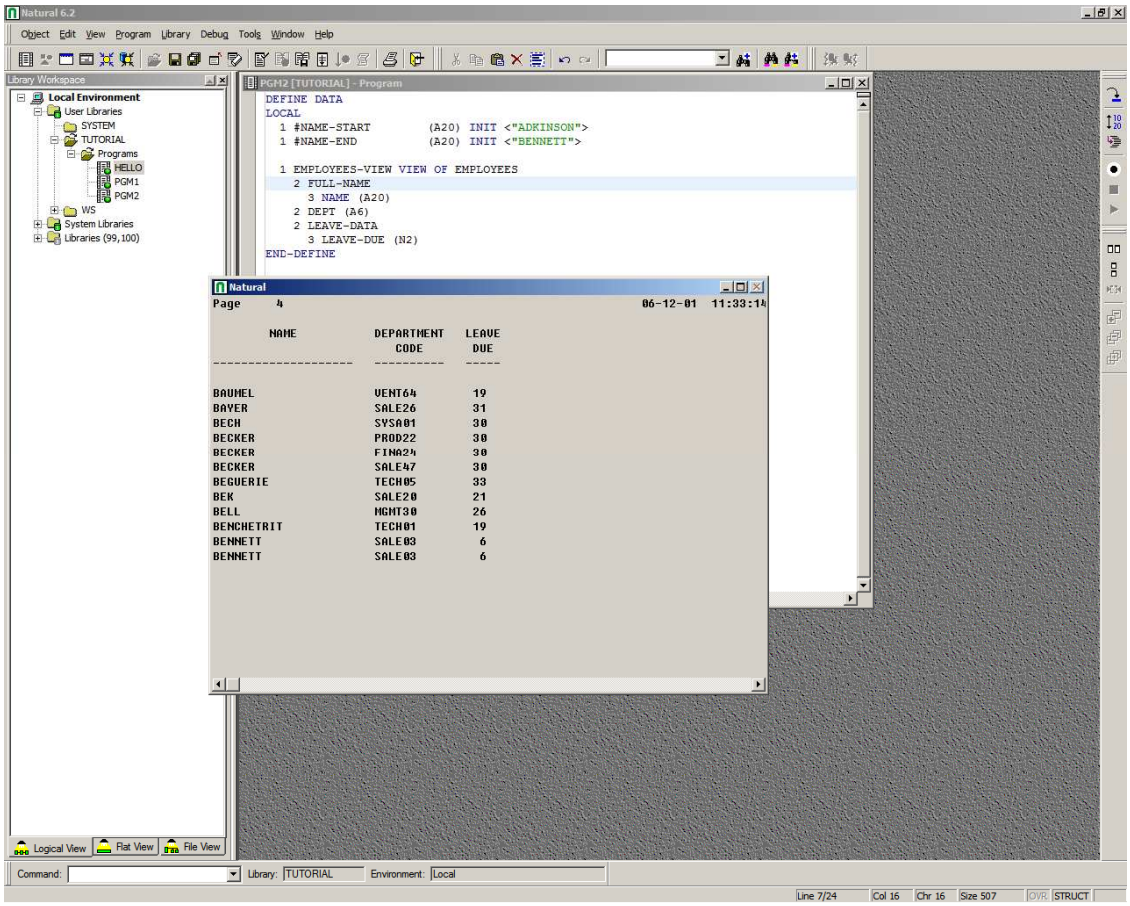

Figure 16 - Output using start and end values

We now have the basics of a database search program. What is missing is how to prompt the user for data, i.e., get the starting and ending name dynamically via screen input.

We modify your program so that input fields for the starting name and ending name will be shown in the output screen. This is done using the `INPUT` statement. So insert the following below `END-DEFINE`:

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

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

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

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

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

The user enters the desired starting name and ending name directly into the `#NAME-START` and `#NAME-END` data fields.

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

If you now run this program you will be asked for some input, Figure 17:

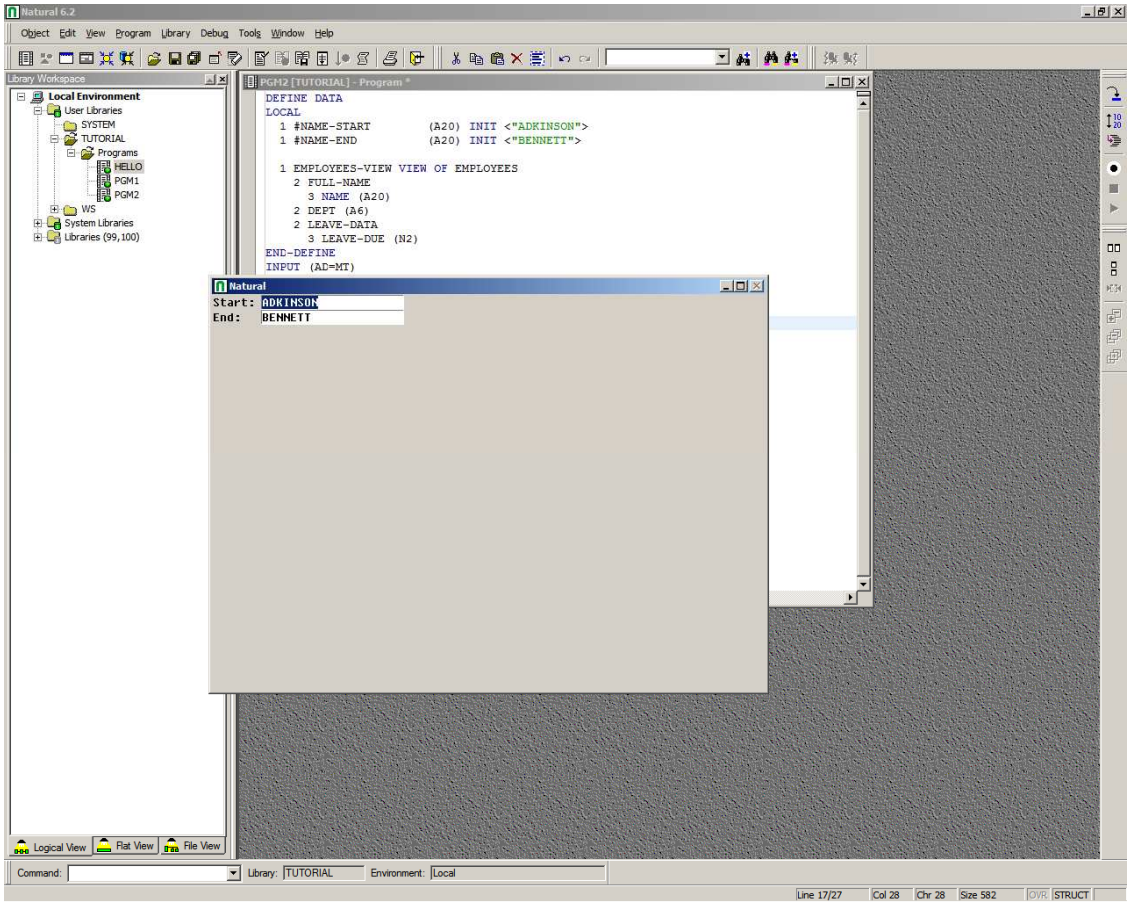

Figure 17 - Restricting output

Simply press ENTER and you will get the same behavior as before.

As it is now, the program terminates after it has shown the requested list of employees. If the user wants a new selection he has to re-start the program.

What we want to do now is allow the user to display a new employees list immediately, without restarting the program. We do this by putting the existing program code into a REPEAT loop.

Insert the following below END-DEFINE:

```
REPEAT
```

REPEAT defines the start of the repeat loop.

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

```
END-REPEAT
```

We now also have to allow the user to terminate the program; we do this by checking for a dot (.) in the input field and, while we are here, we will also allow the user to just input a single value.

So insert the following below the INPUT statement:

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

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

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

The second IF statement checks if the ending value is empty and if so simply moves the start value to the end value field.

Now run the program.

In the resulting output, enter "JONES" in the start field and clear the end field, then press ENTER.

In the resulting list, just the employees with the name Jones are shown.

Press ENTER. Due to the REPEAT loop, the input screen is shown again and you can also see that "JONES" has been entered as the ending name.

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

Stow the program; you will also see that a green dot appears in the icon, indicating that the program has been stowed.

Our program is looking quite good now, but what happens if there are no database records returned? We will now define the message that is to be displayed when the user enters a starting name which cannot be found in the database.

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

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

And insert the following below `END-READ`:

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

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

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

The completed program should now look like this:

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

  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
RP1. REPEAT
INPUT (AD=MT)
  "Start:" #NAME-START /
  "End:  " #NAME-END

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

  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
RD1. READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*

END-READ

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

END-REPEAT
*
END
```

Run the program.

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

The message should now appear at the bottom of the screen, Figure 18.

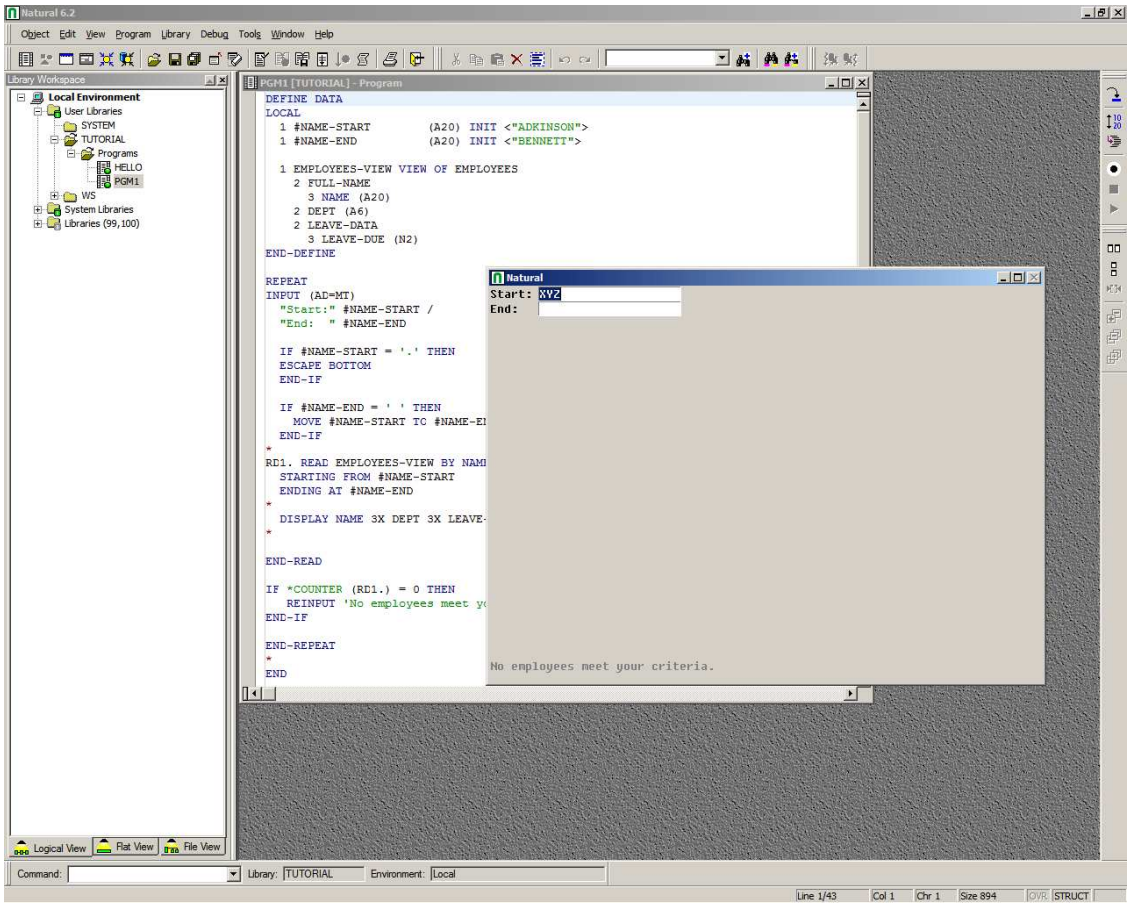

Figure 18 - Error message

Save the program by stowing it and you are finished!!