

Dialog Design

This document tells you how you can design user interfaces that make user interaction with the application simple and flexible:

- Field-Sensitive Processing
*CURS-FIELD and POS(*field-name*)
 - Simplifying Programming
System Function POS
 - Line-Sensitive Processing
System Variable *CURS-LINE
 - Column-Sensitive Processing
System Variable *CURS-COL
 - Processing Based on Function Keys
System Variable *PF-KEY
 - Processing Based on Function-Key Names
System Variable *PF-NAME
 - Processing Data Outside an Active Window
System Variable *COM
 - Copying Data from a Screen
Terminal Commands %CS and %CC
 - Statements REINPUT/REINPUT FULL
 - Object-Oriented Processing
Natural Command Processor
-

Field-Sensitive Processing

*CURS-FIELD and POS(*field-name*)

Using the system variable *CURS-FIELD together with the system function POS(*field-name*), you can define processing based on the field where the cursor is positioned at the time the user presses ENTER.

*CURS-FIELD contains the internal identification of the field where the cursor is currently positioned; it cannot be used by itself, but only in conjunction with POS(*field-name*).

You can use *CURS-FIELD and POS(*field-name*), for example, to enable a user to select a function simply by placing the cursor on a specific field and pressing ENTER.

The example below illustrates such an application:

Example:

```

DEFINE DATA LOCAL
1 #EMP (A1)
1 #CAR (A1)
1 #CODE (N1)
END-DEFINE
*
INPUT USING MAP 'CURS'
*
DECIDE FOR FIRST CONDITION
  WHEN *CURS-FIELD = POS(#EMP) OR #EMP = 'X' OR #CODE = 1
    FETCH 'LISTEMP'
  WHEN *CURS-FIELD = POS(#CAR) OR #CAR = 'X' OR #CODE = 2
    FETCH 'LISTCAR'

```

```

WHEN NONE
  REINPUT 'PLEASE MAKE A VALID SELECTION'
END-DECIDE

END

```

SAMPLE MAP

Please select a function

1.) Employee information _

2.) Vehicle information _ ← **Cursor positioned on field**

Enter code: _

To select a function, do one of the following:

- place the cursor on the input field next to desired function and press ENTER
- mark the input field next to desired function with an X and press ENTER
- enter the desired function code (1 or 2) in the 'Enter code' field and press ENTER

If the user places the cursor on the input field (#EMP) next to Employee information, and presses ENTER, the program LISTEMP displays a list of employee names:

Page 1 2001-01-22 09:39:32

NAME

ABELLAN

ACHIESON

ADAM

ADKINSON

AECKERLE

AFANASSIEV

AFANASSIEV

AHL

AKROYD

Simplifying Programming

System Function POS

The Natural system function POS(*field-name*) contains the internal identification of the field whose name is specified with the system function.

`POS(field-name)` may be used to identify a specific field, regardless of its position in a map. This means that the sequence and number of fields in a map may be changed, but `POS(field-name)` will still uniquely identify the same field. With this, for example, you need only a single `REINPUT` statement to make the field to be `MARKED` dependent on the program logic.

Note:

The values of `*CURS-FIELD` and `POS(field-name)` serve for internal identification of the fields only. They cannot be used for arithmetical operations.

Example:

```
...
DECIDE ON FIRST VALUE OF ...
  VALUE ...
    COMPUTE #FIELDX = POS(FIELD1)
  VALUE ...
    COMPUTE #FIELDX = POS(FIELD2)
  ...
END-DECIDE
...
REINPUT ... MARK #FIELDX
...
```

Full details on `*CURS-FIELD` and `POS(field-name)` are described in the Natural System Variables and System Functions documentation.

Line-Sensitive Processing

System Variable `*CURS-LINE`

Using the system variable `*CURS-LINE`, you can make processing dependent on the line where the cursor is positioned at the time the user presses `ENTER`.

Using this variable, you can make user-friendly menus. With the appropriate programming, the user merely has to place the cursor on the line of the desired menu option and press `ENTER` to execute the option.

The cursor position is defined within the current active window, regardless of its physical placement on the screen.

Note:

The message line, function-key lines and statistics line/infoline are not counted as data lines on the screen.

The example below demonstrates line-sensitive processing using the `*CURS-LINE` system variable. When the user presses `ENTER` on the map, the program checks if the cursor is positioned on line 8 of the screen which contains the option "Employee information". If this is the case, the program that lists the names of employees `LISTEMP` is executed.

Example:

```
DEFINE DATA LOCAL
1 #EMP (A1)
1 #CAR (A1)
1 #CODE (N1)
END-DEFINE
*
INPUT USING MAP 'CURS'
*
DECIDE FOR FIRST CONDITION
  WHEN *CURS-LINE = 8
    FETCH 'LISTEMP'
```

```

WHEN NONE
  REINPUT 'PLACE CURSOR ON LINE OF OPTION YOU WISH TO SELECT'
END-DECIDE
END

```

Company Information

Please select a function

[] 1.) Employee information

2.) Vehicle information

Place the cursor on the line of the option you wish to select and press
ENTER

The user places the cursor indicated by [] on the line of the desired option and presses ENTER and the corresponding program is executed.

Column-Sensitive Processing

System Variable *CURS-COL

The system variable *CURS-COL can be used in a similar way to *CURS-LINE described above. With *CURS-COL you can make processing dependent on the column where the cursor is positioned on the screen.

Processing Based on Function Keys

System Variable *PF-KEY

Frequently you may wish to make processing dependent on the function key a user presses.

This is achieved with the statement SET KEY, the system variable *PF-KEY and a modification of the default map settings (Standard Keys = "Y").

The SET KEY statement assigns functions to function keys during program execution. The system variable *PF-KEY contains the identification of the last function key the user pressed.

The example below illustrates the use of SET KEY in combination with *PF-KEY.

Example:

System Variable *COM

As stated above, only **one** window is active at any one time. This normally means that input is only possible within that particular window.

Using the *COM system variable, which can be regarded as a communication area, it is possible to enter data outside the current window.

The prerequisite is that a map contains *COM as a modifiable field. This field is then available for the user to enter data when a window is currently on the screen. Further processing can then be made dependent on the content of *COM.

This allows you to implement user interfaces as already used, for example, by Con-nect, Software AG's office system, where a user can always enter data in the command line, even when a window with its own input fields is active.

Note that *COM is only cleared when the Natural session is ended.

Example Usage of *COM

In the example below, the program ADD performs a simple addition using the input data from a map. In this map, *COM has been defined as a modifiable field (at the bottom of the map) with the length specified in the AL field of the Extended Field Editing . The result of the calculation is displayed in a window. Although this window offers no possibility for input, the user can still use the *COM field in the map outside the window.

Program ADD:

```

DEFINE DATA LOCAL
1 #VALUE1 (N4)
1 #VALUE2 (N4)
1 #SUM3 (N8)
END-DEFINE
*
DEFINE WINDOW EMP
  SIZE 8*17
  BASE 10/2
  TITLE 'Total of Add'
  CONTROL SCREEN
  FRAMED POSITION SYMBOL BOT LEFT
*
INPUT USING MAP 'WINDOW'
*
COMPUTE #SUM3 = #VALUE1 + #VALUE2
*
SET WINDOW 'EMP'
INPUT (AD=0) / 'Value 1 +' /
              'Value 2 =' //
              ' ' #SUM3
*
IF *COM = 'M'
  FETCH 'MULTIPLY' #VALUE1 #VALUE2
END-IF
END

```

```

Map to Demonstrate Windows with *COM

CALCULATOR

Enter values you wish to calculate

Value 1: 12__
Value 2: 12__

+-Total of Add-+
!               !
! Value 1 +     !
! Value 2 =     !
!               !
!           24  !
!               !
+-----+

Next line is input field (*COM) for input outside the window:

```

In this example, by entering the value "M", the user initiates a multiplication function; the two values from the input map are multiplied and the result is displayed in a second window:

```

Map to Demonstrate Windows with *COM

CALCULATOR

Enter values you wish to calculate

Value 1: 12__
Value 2: 12__

+-Total of Add-+
!               !
! Value 1 +     !
! Value 2 =     !
!               !
!           24  !
!               !
+-----+

+-----+
!               !
! Value 1 x     !
! Value 2 =     !
!               !
!           144 !
!               !
+-----+

Next line is input field (*COM) for input outside the window:
M

```

Positioning the Cursor to *COM - the %T* Terminal Command

Normally, when a window is active and the window contains no input fields (AD=M or AD=A), the cursor is placed in the top left corner of the window.

With the terminal command %T*, you can position the cursor to a *COM system variable outside the window when the active window contains no input fields.

By using %T* again, you can switch back to standard cursor placement.

Example:

```

...
INPUT USING MAP 'WINDOW'
*
COMPUTE #SUM3 = #VALUE1 + #VALUE2
*
SET CONTROL 'T*'
SET WINDOW 'EMP'
INPUT (AD=0) / 'Value 1 +' /
                'Value 2 =' //
                ' ' #SUM3
...

```

Copying Data from a Screen

Below is information on:

- Terminal Commands %CS and %CC
- Selecting a Line from Report Output for Further Processing

Terminal Commands %CS and %CC

With these terminal commands, you can copy parts of a screen into the Natural stack (%CS) or into the system variable *COM (%CC). The protected data from a specific screen line are copied field by field.

The full options of these terminal commands are described in the Natural Terminal Commands documentation.

Once copied to the stack or *COM, the data are available for further processing. Using these commands, you can make user-friendly interfaces as in the example below.

Selecting a Line from Report Output for Further Processing

In the following example, the program COM1 lists all employee names from Abellan to Alestia.

Program COM1:

```

DEFINE DATA LOCAL
1 EMP VIEW OF EMPLOYEES
  2 NAME(A20)
  2 MIDDLE-NAME (A20)
  2 PERSONNEL-ID (A8)
END-DEFINE
*
READ EMP BY NAME STARTING FROM 'ABELLAN' THRU 'ALESTIA'
  DISPLAY NAME
END-READ
FETCH 'COM2'
END

```

```

Page      1                                2001-01-22  08:21:22

      NAME
-----

ABELLAN
ACHIESON
ADAM
ADKINSON
ADKINSON
ADKINSON
ADKINSON
ADKINSON
ADKINSON
ADKINSON
ADKINSON
ADKINSON
AECKERLE
AFANASSIEV
AFANASSIEV
AHL
AKROYD
ALEMAN
ALESTIA
MORE

```

Control is now passed to the program COM2.

Program COM2:

```

DEFINE DATA LOCAL
1 EMP VIEW OF EMPLOYEES
  2 NAME(A20)
  2 MIDDLE-NAME (A20)
  2 PERSONNEL-ID (A8)
1 SELECTNAME (A20)
END-DEFINE
*
SET KEY PF5 = '%CCC'
*
INPUT NO ERASE 'SELECT FIELD WITH CURSOR AND PRESS PF5'
*   MOVE *COM TO SELECTNAME
FIND EMP WITH NAME = SELECTNAME
  DISPLAY NAME PERSONNEL-ID
END-FIND
END

```

In this program, the terminal command %CCC is assigned to PF5. The terminal command copies all protected data from the line where the cursor is positioned to the system variable *COM. This information is then available for further processing. This further processing is defined in the program lines shown in **boldface**.

The user can now position the cursor on the name that interests him; when he/she now presses PF5, further employee information is supplied.

```

SELECT FIELD WITH CURSOR AND PRESS PF5                                2001-01-22  08:20:22

      NAME
-----

ABELLAN
ACHIESON
ADAM  ← Cursor positioned on name for which more information is required
ADKINSON
AECKERLE
AFANASSIEV
AFANASSIEV
AHL
AKROYD
ALEMAN
ALESTIA
    
```

In this case, the personnel ID of the selected employee is displayed:

```

Page      1                                                                2001-01-22  08:20:30

      NAME          PERSONNEL
      -----          ID
ADAM          50005800
    
```

Statements REINPUT/REINPUT FULL

If you wish to return to and re-execute an INPUT statement, you use the REINPUT statement. It is generally used to display a message indicating that the data input as a result of the previous INPUT statement were invalid.

If you specify the FULL option in a REINPUT statement, the corresponding INPUT statement will be re-executed fully:

- With an ordinary REINPUT statement (without FULL option), the contents of variables that were changed between the INPUT and REINPUT statement will not be displayed; that is, all variables on the screen will show then contents they had when the INPUT statement was originally executed.
- With a REINPUT FULL statement, all changes that have been made after the initial execution of the INPUT statement will be applied to the INPUT statement when it is re-executed; that is, all variables on the screen contain the values they had when the REINPUT statement was executed.
- If you wish to position the cursor to a specified field, you can use the MARK option, and to position to a particular position within a specified field, you use the MARK POSITION option.

The example below illustrates the use of REINPUT FULL with MARK POSITION.

Example:

```

DEFINE DATA LOCAL
1 #A (A10)
1 #B (N4)
1 #C (N4)
END-DEFINE
*
INPUT (AD=M) #A #B #C
IF #A = ' '
  COMPUTE #B = #B + #C
  RESET #C
  REINPUT FULL 'Enter a value' MARK POSITION 5 IN *#A
END-IF
END

```

The user enters 3 in field #B and 3 in field #C and presses ENTER.

#A	#B	3	#C	3
----	----	---	----	---

The program requires field #A to be non-blank. The REINPUT FULL statement with MARK POSITION 5 IN *#A returns the input screen; the now modified variable #B contains the value 6 (after the COMPUTE calculation has been performed). The cursor is positioned to the 5th position in field #A ready for new input.

Enter name of field					
#A	_	#B	6	#C	0
	↑	Cursor positioned to 5th position in field			
Enter a value					

This is the screen that would be returned by the same statement, without the FULL option. Note that the variables #B and #C have been reset to their status at the time of execution of the INPUT statement (each field contains the value 3).

#A	_	#B	3	#C	3
----	---	----	---	----	---

Object-Oriented Processing

Natural Command Processor

The Natural Command Processor is used to define and control navigation within an application.

The Natural Command Processor consists of two parts: a **development part** and a **runtime part**.

- The **development part** is the utility SYSNCP. With this utility, you define commands and the actions to be performed in response to the execution of these commands. From your definitions, SYSNCP generates decision tables which determine what happens when a user enters a command.
- The **run-time part** is the statement PROCESS COMMAND. This statement is used to invoke the Command Processor within a Natural program. In the statement you specify the name of the SYSNCP table to be used to handle the data input by a user at that point.

For further information regarding the Natural Command Processor, see the Natural SYSNCP Utility documentation and the statement `PROCESS COMMAND` as described in the Natural Statements documentation.