# ND Screen Handling System

ND-60.088.03

Norsk Data

# ND Screen Handling System

ND-60.088.03

## *NOTICE*

The information in this document is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this document. Norsk Data A.S assurnes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

# PRINTING RECORD

| Printing | Notes |
|---|---|
| 01/78 | ORIGINAL PRINTING |
| 04/79 | Second Printin g – replaces original printing |
| 08/82 | VERSION 3 |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

ND SCREEN HANDLING SYSTEM
Publication No. ND-60.088.03

NORSK DATA A.S
P.O. Box 4, Lindeberg gård
Oslo 10, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the ND Bulletin and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms, together with all types of inquiry and requests for documentation should be sent to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

# PREFACE

## *THE PRODUCT*

This manual describes version 03 of the ND Screen Handling System (NSHS). This is registered in the ND Software Library as part of ND-numbers 10013F,G/10053F. NSHS is delivered on two diskettes containing following files

Diskette 1: I
(ND-10013I-PART1:FLOPPY-USER)SCREEN-DEF-2155I:BPUN;1 I

(ND-10013I-PART1:FLOPPY-USER)SCREEN-LIB-2154K:BRF;1 K
(ND-10013I-PART1:FLOPPY-USER)SCREEN-RTL-2156K:BRF;1 K

Diskette 2: I
(ND-10013I-PART2:FLOPPY-USER)SCREEN-COP-2446A:BPUN;1
(ND-10013I-PART2:FLOPPY-USER)SCREEN-UCO-2447B:SYMB;1 B
(ND-10013I-PART2:FLOPPY-USER)SCREEN-DEM-2181C:SYMB;1

Diskette 3:
(ND-10013I-PART1:FLOPPY-USER)SCREEN-1BANK-A:BRF
(ND-10013I-PART1:FLOPPY-USER)SCREEN-2BANK-A:BRF
(ND-10013I-PART1:FLOPPY-USER)SCREEN-1REEN-A:BRF

Diskette 1 contains the main functions, diskett 2 is used for additional service programs.

## *THE READER*

This manual should be of interest to anybody supervising data processing under NSHS and particularly programmers writing application programs using NSHS. The first chapter, Introduction to NSHS, provides introductory information for anybody wanting a brief description of the system and how it works.

Prerequisite knowledge: Ample experience of Fortran or Cobol is required as application programs are written in one of these languages. Some knowledge of data representation is also required. Cobol programmers must know some Fortran as the subroutine for user control, UCONT, can only be written in Fortran. Examples of application programs and UCONT are given in the User Guide Section.

The manuals

NORD-10/100 FORTRAN System Reference Manual ND-60.074
NORD COBOL Reference Manual ND-60.144

deal with programming Fortran or Cobol on ND computers: they outdate previous manuals on this subject.

## *THE MANUAL*

This manual has been thoroughly revised to coincide with the release of the new NSHS in 1982. The contents have been rearranged and new chapters added. The first chapter is an introduction which in being simply should be comprehensible to all potential users. Then follow three main chapters, Defining the Pictures, Application Programming and a User Guide. The User Guide is a new section containing examples, practical hints for programmers and outlines of special procedures.
The first part of NSHS (creating pictures) is also used by the Data Entry System, ND-60.101.

# TABLE OF CONTENTS

+ \+ \+

*Section:* *Page:*

*Section:* *Page:*

*Section:* *Page:*

*Section:* *Page:*

*Appendix:*

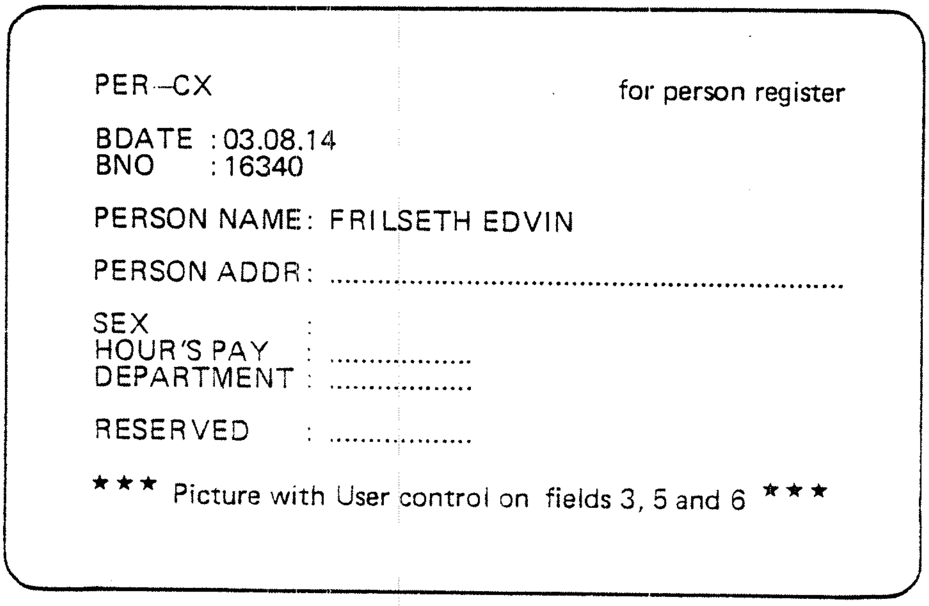
# 1 INTRODUCTION TO NSHS—THE CONCEPT OF THE SYSTEM

NSHS consists of two main parts, the PICTURES and the PROGRAMS. It is used to enter data via a terminal to record files or a database, or to display data records from files or a database on the terminal screen.

*THE PICTURE*

This is the basic element and should be created first. It consists of leading text which is always the same for the same picture, followed by datafields.

The user can design a picture to fit his particular requirements. Eg. it may be practical to have a picture resembling the source document. Data entered at the terminal will automatically appear on the screen in the right position, easily identified by the operator.



```
PER-CX                                       for person register

BDATE : 03.08.14
BNO   : 16340

PERSON NAME: FRILSETH EDVIN

PERSON ADDR: .......................................................

SEX         :
HOUR'S PAY  : .................
DEPARTMENT  : .................

RESERVED    : .................

*** Picture with User control on  fields 3, 5 and 6 ***
```

Figure 1.1 shows a picture for registration of person records, where the first three fields have been filled in and the others left empty. The datafields are indicated by dots. In this case there is only one datafield on each line, but the number of datafields on a line is not restricted to one.

Automatic data control conditions can be defined when a picture is created. If unacceptable data are entered, they are rejected, the terminal gives an audible signal and an error message on the last line of the screen. The operator can then continue with correct data.

A number of standard control conditions can be specified for each datafield on the screen. However the user may want a more sophisticated data check, for instance on relations between different datafields. This is called user control and requires the writing of a special program. User control is called from the picture when a datafield with control number = 9 is encountered. It is possible to have one picture without control and another with, or even to have several pictures apparently the same but with different control on the datafields.

*THE PROGRAMS*

The whole process is directed by the application programs. The program calls the right picture to the screen and transfers data to a record file or database. It should perform the main processing tasks but leave data checking to the system or user control. In this way all processing can be carried out by one application program and one user control program.

Several standard calls can be used in the programs. One call, GTPIC, is of special importance and always appears early in the program. It gives the picture filename and picture name, and then the system has all the necessary information concerning the picture and datafields. In a series of calls, parameters generated from one call are by default used as input parameters to another call. This technique is demonstrated in the User Guide section, which gives practical examples demonstrating the procedure from start to finish ie. running the program. There are also examples of bit/byte manipulation, details of those field definitions which have proved difficult in the past and details of points where the procedures for Cobol and Fortran programs are different.

One of the more sophisticated features of NSHS is that at runtime pictures can be used as PRIVATE or PUBLIC. In most cases private use is satisfactory, but if many users are accessing the same picture at the same time, the system works more efficiently with public pictures, because of the reentrant features of the SINTRAN operating system.

We hope this introduction has explained the basic idea of what NSHS is and how it can be used, and that it will help you as you read on. We wish you success in your further studies!

## 1.1 GENERAL TERMS

Every software system has its own vocabulary of frequently used words and phrases. The most important in the NSHS are as follows:

*Leading text*

This provides information or explanation of a picture. It is defined when a picture is created, is a permanent part of the picture and cannot be modified by an application program at runtime.

*Field*

A position in a picture reserved for data input/output, defined when a picture is created. Fields can be used to display output from a user's program, or to display characters read in from the keyboard, or both.

*Field Characteristics*

These are defined in Screen-Def and include how a field is handled, its maximum length in characters, its format, its edit and control parameters.

*Picture*

The combination of leading texts and fields, defined in Screen-Def. A picture has an 8 character (max.) name which must be unique within one picture file or within any set of pictures used simultaneously in one application. A picture cannot contain more than 255 fields.

## 1.2 PICTURE FILES

Once a picture is defined it is called the source picture.and is stored in a contiguous SINTRAN file with the default filetype designation :PICT. Up to 49 pictures identified by their individual picture names can be stored on this file. Before a picture is used by an application program, it must be compiled into a runtime format. A compiled picture, called the object picture, is stored on another contiguous SINTRAN file with default filetype designation (:OBJ). This file may also contain up to 49 pictures. The NORD File System manual contains a more detailed discussion of file handling.

# 2 THE SCREEN DEFINITION SYSTEM (Screen-Def)

## 2.1 *GENERAL*

The interactive utility program for creating and manipulating pictures is activated by the SINTRAN command

@SCREEN-DEFINITION

This is the name the program was given when it was loaded and can be abbreviated if this does not produce an ambiguous name.

A heading page appears and you are asked

Terminal type?*

The different possibilities are displayed on the screen and you answer with the integer number corresponding to your terminal. READY appears on completion of all operator commands. Typing a space or carriage return sets the program in command-receiving mode. A command can be entered when

SPM*

(Screen Picture Maintenance) appears on the screen.

Available commands are listed below and described on the following pages. All commands can be abbreviated.

*If the terminal type is known to SINTRAN, this information will be used by NSHS and the start procedure will be omitted.

AVAILABLE COMMANDS

| | |
|---|---|
| Picture commands | DEFINE-PICTURE-SIZE |
| | CREATE-PICTURE |
| | COMPILE-FILE |
| | |
| | DESCRIBE-PICTURE |
| | DESCRIBE-FILE |
| | DUMP-PICTURE-FILE |
| | |
| | MODIFY-PICTURE |
| | DELETE-PICTURE |
| | SCRATCH-PICTURE-FILE |
| | |
| | RESCUE-PICTURE |
| | CLOSE-FILES |
| General commands | HELP |
| | SET-TERMINAL-TYPE |
| | SET-DISPLAY-MODE |
| | EXIT |
| Data Entry commands | SET-MUST-READ |
| | RESET-MUST-READ |
| | |
| | SET-VERIFY |
| | RESET-VERIFY |

## 2.2 *PICTURE COMMANDS*

If no files have been opened before a picture is entered, the system will ask for source and object files. Both file name and type must be given. If the files which have been opened are not source/object files, you will be asked if you wish to continue. If you give Y and carriage return, these files will then be defined as source/object files.Typing any other character will result in a return to command input mode.

NOTE: Both the source and object picture files must be contiguous. Otherwise an error message appears on the terminal and control is returned to command input mode (see 1.2).

### 2.2.1 *DEFINE-PICTURE-SIZE*

This command specifies the number of lines (between 1 and 31) on the display and the number of characters (between 1 and 127) per line.

A picture can have a maximum of 31 lines each containing 127 characters. But most display terminals have only 24 lines with 80 characters. (The Tandberg TDV 2100 has 25 lines with a maximum line length of 80 characters.) Pictures with more than 24 lines can be written to display terminals but the additional lines will not be shown immediately. (NSHS never uses column 80 of a VDU and the last line on a VDU is used for system purposes, so default picture size is 23 lines of 79 characters.)

If the number of lines in the picture exceeds the line-capacity of the terminal, the picture is automatically divided into two pages. The second page is automatically displayed when the cursor is moved beyond the first page. If it is moved back the pages are switched back again.

If the line contains more than 79 characters, the line is split into two lines. Pictures defined with more than 79 characters per line can only be used in the WRITE-PICTURE-TO-FILE (WPRTF) command.

### 2.2.2 *CREATE-PICTURE*

This command clears and formats the display and sets the cursor in the home position (top left). A picture can then be defined by entering leading texts, headings and field descriptions. Leading texts, preceded by display mode characters, or positioning spaces are entered directly. Fields are defined by pressing control/E. Editing facilities are described later in the manual.

When a picture is completed, type control/W and it will be stored and compiled. If you do not want to keep the picture, type control/F.

### 2.2.3 *COMPILE-FILE*

This command compiles pictures in the source file and writes them to the object file. Old object pictures are replaced or new pictures added. This command is not normally necessary as pictures are compiled automatically on termination of CREATE-PICTURE and MODIFY-PICTURE commands. It is useful if the object picture file is destroyed by accident. A single modified picture must be compiled using MODIFY-PICTURE, immediately followed by control/W.

### 2.2.4 *DESCRIBE-PICTURE*

This command generates a description of a picture containing the picture layout, object picture size, number and description of defined fields. This is written to a user selected file.

### 2.2.5 *DESCRIBE-FILE*

This command is equivalent to DESCRIBE-PICTURE, except that all the pictures in the source file are described.

## 2.2.6 *DUMP-PICTURE-FILE*

This command writes to a specified list file the number of blocks on a picture file, a table showing which blocks are occupied and the names of the pictures actually stored on the file.

Then the following text is displayed
NR OF BLOCKS ON FILE:  40  IF NOT OK GIVE NEW NR, ELSE 0:  20

NEW NR refers to the number of blocks to be listed, starting from 0. In this case the first 24 blocks will be listed because $20_{10}$ is converted to $24_8$.

Complete information for source and object file is given separately.

## 2.2.7 *MODIFY-PICTURE*

After this command you are asked for the name of the picture to be modified. If found, this picture will be displayed on the screen and the cursor returned to the home position. The picture can be modified using the same editing and control facilities as in CREATE-PICTURE.

## 2.2.8 *DELETE-PICTURE*

This command can be used to display the names of pictures on the picture files and to delete these pictures.

Names of pictures on the *source* picture file are displayed first with the question which picture is to be deleted. Give the name of the picture to be deleted or CR if no deletion is required.

Names of pictures on the *object* picture file are then displayed with the question which picture is to be deleted. Give the name of the picture, or CR to return to SPM*.

It is possible to use pictures which only exist on the object file, but some commands, eg.SPM*DESCR-PIC, SPM*MODIFY-PIC, cannot be used. Pictures missing from the object file can be easily reestablished by the command SPM*MODIFY-PIC (single picture) or SPM*COMPILE-FILE (whole file).

### 2.2.9 *SCRATCH-PICTURE-FILE*

**This command destroys source and object picture files. Source file, object file, or both, can be scratched. The system asks for confirmation of the initial decision.**

### 2.2.10 *RESCUE-PICTURE*

When a source picture has been created or modified and control/W pressed, if the source file is full the picture data remains in core. This command writes the picture back to the original source file, or if desired to another source file.

The procedure is as follows:

Give the command

```
    SPM*CLOSE-FILES.
```

Try to expand the files with the SINTRAN command:

```
    SPM*@EXP-FILE _ _ _ _ .
```

If this does not succeed, new files must be created:

```
    SPM*@CREATE-FILE  NAME:PICT,N
    ØPM*@CREATE-FILE  NAME:OBJ,N
```

where the number of SINTRAN pages is subtituded for N.

Now give the command:

```
    SPM*RESCUE-PICTURE
```

### 2.2.11 *CLOSE-FILES*

This command closes the source and object files.

## 2.3 *GENERAL COMMANDS*

### 2.3.1 *HELP*

A list of all available commands is written on the terminal.

### 2.3.2 *SET-TERMINAL-TYPE*

Available terminal types are listed on the screen and you answer with the integer number corresponding to your terminal. This command is automatically issued at the beginning of the Screen Definition System, but if the terminal type is known to SINTRAN it will be used by NSHS and this procedure omitted.

### 2.3.3 *SET-DISPLAY-MODE*

Most terminals require a position on the VDU for representing the control-code for blink/inverse video/underline and so forth. Therefore, default display mode is set to 1 which indicates that before and after each field a position is reserved for display control-code. The control code for Normal is automatically defined after each field. When defining a leading text in a special display mode, a position on the VDU is reserved.

Another possibility is:

0 : Particularly used for terminals where the display-code does not require a VDU position. (No position is reserved for the display on the VDU).

2 : Leading text can be defined with special display, and a position will be reserved for the display-code. For fields, no position for SET-DISPLAY is reserved, and fields can only be used with normal display. In this mode fields can be 'packed' (no spaces between fields).

### 2.3.4 *EXIT*

This command stops the picture definition module and returns control to the operating system.

## 2.4 *DATA ENTRY COMMANDS*

The following commands are used in connection with the Data Entry system. They are used to set/reset the MUST-READ or VERIFY features on specified fields. When a picture is modified the commands must be repeated for each field.

These commands have no effect on pictures directed by application programs.

### 2.4.1 *SET-MUST-READ*

It is possible to have MUST-READ on individual fields. If this command is given you must indicate whether a field is to be a mandatory read.

### 2.4.2 *RESET-MUST-READ*

This command reverses the action taken in SET-MUST-READ.

### 2.4.3 *SET-VERIFY*

It is possible to have individual verification of fields. If this command is given you must indicate for each field whether it is to be verified.

### 2.4.4 *RESET-VERIFY*

This command reverses the action taken in SET-VERIFY.

## 2.5 PICTURE CREATION AND MODIFICATION

### 2.5.1 Editing Facilities

The editing facilities used in picture creation and modification operate on one line at a time, allowing characters to be copied from the line above, from the old line or input through the keyboard. The following control characters are used:

| | | |
|---|---|---|
| Control/A: | (←) | remove previous character or field |
| Control/C: | (→) | copy one character from old line |
| Control/D: | (↓) | copy remainder of old line from current position up to and including the last character |
| Control/P: | | copy one character from previous line |
| Control/Q: | | delete characters on this line and return cursor to position one on the same line |
| Control/R: | (↑) | copy previous line up to and including last character |

Line editing is normally terminated by carriage return (CR). When in a line, ← is equivalent to Control/A, → to Control/C. Beyond position one, ↑ is equivalent to Control/R and ↓ to Control/D, but the arrows also terminate the line.

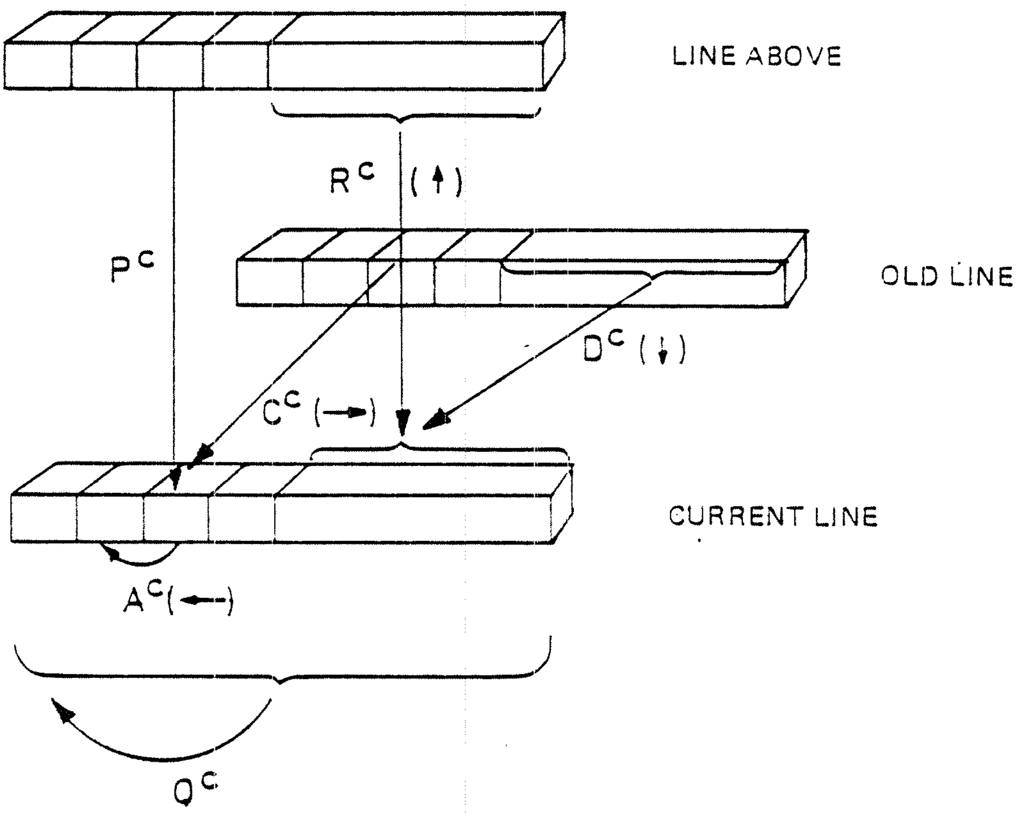
If Control/A,C,P,← or → are typed when a line is being edited, a field is handled as one character. Eg. Control/A immediately after a 10 character field replaces the 10 characters by spaces and moves the cursor back 10 places. When a line has been edited CR should be used to move the cursor to the first position of the next line. The cursor can then be moved by ↑, ↓ or  . CR always deletes all characters on the line to the right of and including the cursor position.

Control/J: insert one line above the current line; this and all lines below are pushed down one line and the last line is lost.

Control/G: remove the current line; all lines below are raised by one line and the last line becomes an empty line.

These functions only work when the cursor is at the beginning of a line. They are IRREVERSIBLE.

The previous field, or any field already defined on the line to the left of the cursor can be repeated on the same line by typing ≤. (See Fig. 2.1.)



$J^C$ = INSERT NEW LINE

$G^C$ = REMOVE CURRENT LINE

↖ = COPY PREVIOUS FIELD

WHEN CURSOR IN COLUMN 1.

↑<br>
↓ } MOVE CURSOR<br>
↖

*Figur 2.1: Editing Characters*

## 2.5.2 *Creating Pictures*

When the initial procedure described in Section 2.1 is completed, the command sign SPM* appears at the top left of the screen.

To create a new picture use
SPM*DEF-PIC-SIZ to limit picture size to part of the screen; if this is not used the whole screen is defined by default.

or directly
SPM*CRE-PIC
if the whole screen can be used.

After the command press CR and further questions are asked or advice given. The screen is cleared and the cursor placed in the home position.

The picture can now be edited by moving the cursor (by pressing the arrow keys) to the position where the leading text, which may be the heading, is required. Text printed on the keyboard and displayed is automatically taken as leading text. Control/U underlines the text. (Control/N terminates underlining.) Any editing character takes up one position, so it is recommended to start text or datafields in position two so that columns can be lined up.

The first datafield after the leading text is defined by positioning the cursor where the field should start and pressing Control/E. The cursor is moved to the last screen line and questions asked about storage, edit, fill and sign suppress codes, thus defining the field. (See Section 2.6.4.) When the last question is answered the cursor moves back onto the screen and new leading texts or fields can be defined.

Details of editing are given below and you are also refered to the User Guide section.

## 2.5.3 *Definition of Display Modes*

If the VDU terminal permits, leading texts can be displayed with blink, low intensity etc. The definition of Display Mode is independent of terminal type and is achieved by typing one of the following control characters:

| | |
|---|---|
| Control/B | Blink |
| Control/L | Low intensity |
| Control/N | Normal |
| Control/O | Invisible |
| Control/U | Underline |
| Control/V | Inverse video |

A display mode is terminated by giving Normal, giving a different display mode character, by terminating the line or by defining a field.

## 2.5.4 *Definition of Fields*

When a source picture is being created or modified, typing control/E indicates that a field is to be defined. The system then asks:

| *Question* | *Possible Answer* |
|---|---|
| Storage code | 1-5 |
| Edit code | 0-9, A-S |
| Fill code | 0-1 |
| Sign suppress code | 0-1 |
| Number of significant characters in the field | actual number of characters |

If the answers are acceptable the edited field is filled with 9, A or X depending on the edi\ code. The system then asks

| | |
|---|---|
| Field control functions | 0-10 |

These codes and control functions are explained in the following sections.

### 2.5.4.1 Storage Code

Storage code defines the internal representation of characters read in.

*Storage code:*

1. Single integer (maximum 5 digits-32768 ≤ value ≤ 32767).
2. Double integer (maximum 10 digits-2 147 483 648 ≤ value ≤ 2 147 483 647).
3. Byte (maximum 79 characters).
4. FORTRAN IV. Each 16 bit word contains a 4 digit integer corresponding to 4 decimal digits. Maximum 77 digits.
5. Binary coded decimal. Each 16 bit word represents 4 decimal digits, each digit a binary value. Maximum 79 digits.

Hints:

For fields to be used by the Data Entry Compiler use 3.
Numeric fields to be used by Cobol application programs use 1, 2 and 3 only.
All fields to be used by Data Entry 2 should be 3.
te that there is no 'real' ('floating') option.

The User Guide section contains a further explanation of storage codes.

### 2.5.4.2 Edit Codes

Edit codes define how fields are presented on the video screen. The formats can be changed within certain limits to suit the needs of the user. (See Appendix A.)

*Numeric fields 1*
EDIT CODE 0-9.
For numeric fields with a decimal point. The number given indicates the number of digits to the right of the decimal point. The integer part of the numeric is separated by dots into groups of three digits eg. 2.345.678,1234

Permitted input characters:

digits 0-9
comma (,)
full stop (.)
minus (-)

The minus sign can be typed before or after the number. Full stop and comma can be used for decimal point, but the decimal point displayed on the screen is (,). The grouping separator is (.). This can be changed by patching. (See Appendix A.)

*Numeric fields 2*

The codes and their effects are as follows:

A digits right justified, no editing characters
B edited as Norwegian bank account number (eg. 8360.21.06325)
C edited with dots between each two digits, used for clock and date (eg. 10.02.81)
F digits left justified, no editing characters

Permitted input characters:
0-9 and minus (—).

*String fields*

String fields must have storage code 3 (byte). The codes and their effects are as follows:

| | |
|---|---|
| K | numeric string; left justified |
| L | alphabetic string; left justified |
| M | alphnumeric string; left justified |
| N | numeric string; with edit characters |
| O | alphabetic string; with edit characters |
| P | alphanumeric string; with edit characters |
| Q | numeric string; right justified |
| R | alphabetic string; right justified |
| S | alphanumeric string; right justified |

When fields are defined the field positions are filled by:

| | |
|---|---|
| 9 | for numeric fields or strings |
| A | for alphabetic strings |
| X | for alphanumeric strings |

and with editing characters and positioning appropriate to the given code.

### 2.5.4.3 Fill Code

Fill code defines how unused character positions are filled on the VDU.

*Fill code:*

| | |
|---|---|
| 0 | spaces |
| 1 | zeros |

This code affects the screen and also the record layout for left justified fields. (See Appendix A.) Cobol records require leading positions to be filled with 0; this is achieved automatically by the Cobol setting in ITERM 3. (See Section 3.3.3.)

### 2.5.4.4 Sign Suppress Code

If values contained by a digit field are always positive, the sign position to the right of the field can be discarded.

*Sign suppress code:*

| | |
|---|---|
| 0 | field has sign position |
| 1 | field has no sign position |

### 2.5.4.5 Number of Significant Characters

This is the maximum number of digits that can be typed into a field; editing characters must be taken into account.

### 2.5.5 *Field Control Functions*

The control functions must be defined when the picture is created. They apply to the check-record function and to the reading of fields. If more than one control function is required for a field, combined control (10) must be given initially.

*Control code:*

| | |
|---|---|
| 0 | no control |
| 1 | default value |
| 2 | legal values |
| 3 | illegal values |
| 4 | legal range |
| 5 | illegal range |
| 6 | add field/accumulation field |
| 7 | equivalent field |
| 8 | system defined control algorithm |
| 9 | user defined control algorithm |
| 10 | combined control |

### 2.5.5.1 Default Value

*Control code = 1*

This is defined when a picture is created. If the operator gives a terminating character (eg. CR) without any other character, the field is given the default value.

### 2.5.5.2 Legal Values

*Control code = 2*

The number of legal values and the values themselves are defined when a picture is created. The values read into the field are checked against those in the legal values table. If no match is found, a message is given and the cursor repositioned at the start of the field.

### 2.5.5.3 Illegal Values

*Control code = 3*

The number of illegal values and the values themselves are defined when a picture is created. Values are only accepted if they differ from those in the illegal values table.

### 2.5.5.4 Legal Range

*Control code = 4*

The legal range is defined when a picture is created. Values are only accepted if they are within or equal to the given limits.

### 2.5.5.5 Illegal Range

*Control code = 5*

The illegal range is defined when a picture is created. Values are only accepted if they are outside the given range.

### 2.5.5.6 Add Field

*Control code = 6*

The value read into this field is added to another field, the accumulation field. An add-field may also be an accumulation field, but in this case values cannot be read into the field. (See the following section.)

### 2.5.5.7 Accumulation Field

*Control code = 6*

Values read into other specified fields are accumulated in this field; values cannot be read in directly. The start value of an accumulation field is zero, or the value represented by its corresponding data element when the read-fields call is entered (if the field read bit is set).

Accumulation fields cannot be read by a RFLDS (read-fields) call, but they can be written into by a WFLDS (write fields) call. They accept values from other accumulation fields. The values of ordinary add fields are added to the accumulation field as they are entered and the new value of the accumulation field is displayed. The value of one accumulation field is not added to another accumulation field until the RFLDS call is terminated.

Any combination of field types is allowed provided there are sufficient significant digits in the accumulation field. If the number of digits entered or accumulated is too great (overflow) the field is filled with asterisks (*).

## 2.5.5.8 Defining Add and Accumulation Fields

The last question when defining fields concerns control. Add and accumulation fields are defined by control code = 6. The procedure is carried out in two steps:

1. 'Field number of accumulating field or 0 if none:'

   Give the number of the field where this field is accumulated. The first data field has the number 1 etc. If the current field is a receiving field only, answer 0.

2. 'Number of fields to be accumulated or 0 if none:'

   Give the number of previously defined fields that are being accumulated in the current field. If the field is an add-field only, answer 0.

If a number of fields are to be accumulated you are asked which fields:

(Next) accumulated field number

Give the first field number and the question is repeated for each add-field being accumulated.

Logical errors are detected when the picture is compiled. The User Guide section contains a further explanation and practical examples.

## 2.5.5.9 Equivalent Fields

*Control code = 7*

If fields in the same picture hold the same data, the data need only be entered in the first field if this code is used.

Eg. in a picture where fields 1, 3, 4 and 6 are identical, field 1 is defined followed by code 7.
You are asked:
    number of equivalent fields : 4
    (next) equivalent field number : 1
The second question is repeated three times and the answers 3, 4 and 6 given.

Equivalent fields must have the same storage code.
The User Guide section has an example of how to define equivalent fields.

### 2.5.5.10 System Defined Program Control

*Control code = 8*

Algorithm control functions in general use such as checking bank account or person numbers can be incorporated into the picture handling system. They are explained in Appendix D.

### 2.5.5.11 User Defined Program Control

*CONTROL CODE = 9*

In this case the user can specify his own control algorithms. For example two or more datafields can be compared and processing continue depending on the result of this operation.

Calls for control are made from the picture and nowhere else. The same picture with different controls on the various fields can be obtained by redefining the picture with different names.

User control is written as a Fortran subroutine with the standard name UCONT and the following parameters:

UCONT (user-control-number, picture-number, field-information-array, number-of-fields, field-number-array, data-record, data-element-index-array, index-to-this-field-number, field-read-bit-array, status)

When a picture is created and User control defined on some field, you are asked to give a User control number. This number is used as the first parameter in UCONT and the first statement in the User control program is therefore frequently a GOTO (branching) to the right part of the program.

The User Control-Number must be a number between 1 and 256. How to use the UCONT-routine, see Section 6.7.6 and Appendix F.

You cannot have more than one UCONT, thus all User control must be put into this subroutine. All kinds of control should be put into the UCONT, and it is then possible to run an application with one Main program and one User control. The standard name for the User control is Subroujtine UCONT(.....) but the file name may be chosen by the user.

The User control is activated from the program call RFLDS (read-fields). The last parameter both in UCONT and the program calls is the STATUS. In UCONT the Status parameter (often abbreviated IST) is used by the programmer to direct the reaction by RFLDS.

If IST=0 On return from UCONT, the check is performed OK and we skip to the next field.

If IST=—1 on return,it means taht the chec was not satisfactory. Usually some error report is programmed to the bottom line of the screen, and the cursor will be positioned at the beginning of the field. The operator may then try with new data.

If some formal error occurs during execution of UCONT, this is reflected by the status parameter inthe call RFLDS chcks the field twice: before the field is read into (IST=—1) and after (IST= +1)

Note that a dummy routine called UCONT is defined in the Screen Library (NSL) always giving status= +1. The user defined UCONT should therefore be loaded before NSL.

For the writing of UCONT several useful subroutined are at disposal. See appendix F.

Examples of UCONT in connection with pictures and application programs are given in the User Guide Section.

### 2.5.5.12 Combined Control

*CONTROL CODE = 10*

This enables checking of field values based on combinations of individual control functions. Combinations are formed by establishing logical AND, OR, AND/OR relationships between the individual functions or groups of functions.

The control functions are divided into four groups:

1. Default Value

2. Legal-Illegal Value
   Legal-Illegal Range
   System Defined Control

3. Add Fields

4. User Control Algorithm

This sequence must be followed when using combined control. Eg. Group 4, User Control Algorithm, must be defined after Group 3, Add Fields. The groups are linked by AND. Within a group functions can be combined in any order. Within Group 2 functions should be combined with AND or OR, not both.

Example:

To define a two digit field with the following control criteria:

1. Values between 10 and 20 not accepted

2. Three digit numbers not accepted

3. Negative yalues not accepted

Input value X must be such that
$0 \leqslant X \leqslant 10$
$20 \leqslant X \leqslant 100$
(Legal range 0—10).OR.(Legal range 20—99)

The Combined Control function does impose a few limitations:

- a maximum of 40 functions may be combined
- for User Control Algorithm
  Equivalent fields
  Default fields (values)
  only one control of each type may be defined
- no Accumulation fields may be defined
- User Control function must always be defined last

An example using Combined Control is given in the User Guide section.

## 2.5.6 *UCONT Routine for Automatic Transfer of Field-values from one Picture to Another*

This routine can also be used by the data entry system.

Using this routine means that when a value in a field is repeated in several consecutive records the value only has to be entered once.

The routine is called when the first record in the sequence is entered. The operator locks the fields that do not require changing. The UCONT routine is activated through the last field in a picture. (The only function of this field is to activate the UCONT routine.)

*FUNCTIONAL DESCRIPTION*

Define the picture:

The last field in the picture is defined as a control-field with the specifications:

| | |
|---|---|
| Storage code | 3 |
| Edit code | L |
| Number of characters | 1 |
| Control number | 9 |
| User control number | 1 |

Use of the routine when entering data:

Enter data to the picture; when the control-field is reached enter L to activate Screen-UCONT. Anything but L means do not activate UCONT.

When screen-UCONT is activated the cursor is placed in the first field in the picture. The following commands are then available:

| | |
|---|---|
| cursor-right | unlock field, jump to next |
| L | lock field, jump to next; locked fields will appear in low intensity if the terminal has this feature |
| cursor-left | move the cursor back one field |
| cursor-up | move the cursor back to, control-field; the question FUNCTION FIELD TO BE LOCKED? (Y or N) is asked |
| Y | lock control-field; it must be unlocked by the application program if it is to be accessed later. (In Data-entry it is unlocked when leaving the APPEND command.) |
| N | control-field remains unlocked; UCONT can be activated for each record entered |

The user must compile the screen-UCONT routine. If the Screen-RT-library is to be used, the UCONT routine must be compiled in reentrant mode.

If the user has his own control routines, they must be incorporated in the UCONT routine above.

```
      SUBROUTINE UCONT(IC,IPN,IFINFO,N,FNA,REC,DEIA,INDEX,IREDBITS,IST)
C
C       THE ROUTINE IS USED FOR AUTOMATIC TRANSFERE OF FIELD VALUES
C       FROM ONE RECORD TO THE NEXT IN A LOOP-SEQUENCE
C       WHERE INPUT-RECORDS ARE READ FROM THE SAME PICTURE.
C
C     WHERE:

C     IC       : THE USER CONTROL NUMBER
C     IPN      : THE PICTURE NUMBER
C     IFINFO   : FIELD INFORMATION ARRAY
C     N        : NUMBER OF FIELDS
C     FNA      : FIELD NUMBER ARRAY
C     REC      : RECORD
C     DEIA     : DATA ELEMENT INDEX ARRAY
C     INDEX    : INDEX TO THIS FIELD NUMBER
C     IREDBITS: BIT ARRAY WITH BIT SET IF FIELD READ
C     IST      : RETURN STATUS


      INTEGER FNA(1),REC(1),DEIA(1),IREDBITS(1),IFINFO(1)
      INTEGER IFINFU(22),PINCH
      ASSEMBLY GFINF,MCURS,PINCH,POUTC,IPNUM

C                                ;% INPUT-STATUS IS -1 BEFORE ANY VALUE IS
C                                ;% GIVEN TO THE FIELD. NO ACTION IS TAKEN.
      IF (IST .EQ. -1) GOTO 90

      IADDR = IPNUM(IPN)
   10 IACT  = DEIA(INDEX)
      IDATA = REC(IACT)

      IF (IC .NE. 1) GOTO 80
C===================================
C     OR MORE GENERAL:            =
C     GOTO (1,2,3,4,.....,255) IC =
C 1   CONTINUE                    =
C       ;% CONTROL 1; GO TO RETURN =
C 2   CONTINUE                    =
C       ;% CONTROL 2; GOTO RETURN =
C """"                             =
C """"                             =
C """"                             =
C 255 CONTINUE                    =
C       ;% CONTROL 255; GOTO RETURN =
C===================================

C                             ;% REST OF THE CODE IS ACTIVATED ONLY IF
C                             ;% USER-CONTROL NUMBER IS 1.
      IDATA = ISHFT(IDATA,-8).AND.177B
C                                       ;% IDATA = INPUT VALUE IN FIELD.
C                                       ;% 111B = "I",114B = "L"
      IF (IDATA .EQ. 111B)    GOTO 70
      IF (IDATA .EQ. 114B)    GOTO 30
C             ;% FOR ALL OTHER VALUES, JUST RETURN.
      GOTO 80

  30  CALL WMSGE('TYPE L IF LOCK REQUIRED,IF NOT,USE:CURSOR RIGHT,IF FIN
     -ISHED, USE CURSOR HOME*')
```

```
      J=0
C                 ;% LOOP FOR TREATING EACH FIELD. THE LOOP INDEX J IS
C                 ;% CALCULATED FROM INPUT VALUES.
   40 J = J + 1
      IF (J .LE. 0) J = 1
      IF (J .GT. N) GOTO 70
C                             ;% HERE J IS BETWEN 1 AND NUMBER OF FIELDS.
C                             ;% GET INFORMATION ABOUT FIELD, PLACE CURSOR
C                             ;% AT BEGINNING OF FIELD, READ ONE CHARACTER.
   45 IFN = FNA(J)
      CALL GFINF(IFINFU,IFN,IADDR)
      CALL MCURS(IFINFU(11),IFINFU(12))
   50 ICHAR = PINCH(1)
C                        ;% CHARACTER READ TO ICHAR.
      IF (ICHAR .EQ. 11B) THEN
C                                    ;% CURSOR RIGHT: RESET EVENTUALLY
C                                    ;% LOCK ON FIELD, WRITE FIELD WITH
C                                    ;% NORMAL DISPLAY, GO TO NEXT FIELD.
         CALL RLOCK(IPN,1,J,FNA,IST)
         CALL WFLDS(0,IPN,1,FNA(J),REC,DEIA(J),IST)
         GOTO 40
      ELSEIF (ICHAR .EQ. 114B) THEN
C                                  ;% L: WRITE FIELD WITH LOW INTENSITY,
C                                  ;% LOCK FIELD, GO TO NEXT FIELD.

         CALL WFLDS(3,IPN,1,FNA(J),REC,DEIA(J),IST)
         CALL ZLOCK(IPN,1,J,FNA,IST)
         GOTO 40

      ELSEIF (ICHAR .EQ. 34B) THEN
C                                    ;% CURSOR UP FOR FINISH.
         GOTO 70
      ELSEIF (ICHAR .EQ. 10B) THEN
C                                   ;% CURSOR LEFT FOR STEP BACK TO
C                                   ;% PREVIOUS FIELD.
         J = J - 2
         GOTO 40
      ENDIF
C                ;% HERE IF ILLEGAL INPUT GIVEN, BELL ON TERMINAL,
C                ;% GO TO NEW INPUT.
      CALL ZBELL
      GOTO 50

   70 CONTINUE
C                ;% FINISH SEQUENCE: WRITE MESSAGE TO LAST LINE OF TERMINAL,
C                ;% PLACE CURSOR ON FUNCTION FIELD, READ ONE CHARACTER.

      CALL WMSGE('FUNCTION FIELD TO BE LOCKED ? (Y OR N)*')
      CALL MCURS(IFINFO(11),IFINFO(12))
      ICHAR = PINCH(1)
C                       ;% CHARACTER READ TO ICHAR.
      IF (ICHAR .EQ. 131B) THEN
C                                 ;% Y FOR YES, ECHO CHARACTER,
C                                 ;% LOCK FUNCTION FIELD.

         CALL POUTC(1,131B)
         CALL ZLOCK(IPN,1,INDEX,FNA,IST)
      ELSEIF (ICHAR .EQ. 10B) THEN
C                                  ;% CURSOR LEFT: GO TO PREVIOUS FIELD.
         J = J - 2
```

```
            GOTO 45
        ELSEIF (ICHAR .EQ. 116B) THEN
C                                         ;% N FOR NO: ECHO CHARACTER, RETURN
            CALL POUTC(1,116B)
        ELSE
C                     ;% ILLEGAL INPUT, TRY AGAIN.
            GOTO 70
        ENDIF

   80   IST = 0
        RETURN

   90   IST = -1
        RETURN
        END
```

## 2.5.7 *Termination of Picture Editing*

Created pictures must be given a name. A modified picture can be written back to the original picture it came from, to another picture (replacing the old one) or to a new picture. Control/W typed at the beginning of a line indicates that the user is satisfied with the screen picture and wishes to store it. After control/W is typed, the picture is automatically compiled to the object file; a list file is requested and any compilation errors are written on the list file.

If a modified picture becomes to large for its current file, an error message is given. The RESCUE-PICTURE command can be used to save the picture.

Control/F indicates that the current screen picture is not to be stored. It produces abort at any time when outside the field definition.

The various error messages are shown in Appendix C.

# 3 APPLICATION PROGRAMS IN NSHS

Using the ND Screen Library System (NSL).

## 3.1 *GENERAL*

As explained in the Introduction, data processing in the Screen Handling System is directed by application programs. These are preferably written in Fortran or Cobol, but any language which can call Fortran subroutines can be used.

In the programs the use of pictures and manipulation of datafields is carried out by calls to the NSHS library. The programs look slightly different in Fortran and Cobol. Eg. the call GTPIC (Get-Picture) is as follows:

```
Fortran:    CALL  GTPIC(PFNS,NOP,PNAS,PNA,IST)
Cobol:      CALL  GTPICC
            USING  PIC-FI-NAM  NO-OF-PIC  PIC-NAM-STR
             -PIC-NO-ARR STATUS
```

Note that the name of the Cobol subroutine has got an extra C.

STATUS is the last parameter in all calls and the value should be checked immediately. So all calls are followed by an IF-statement. (If the value equals 0 the call was carried out satisfactorily.)

Like all other programs, the NSHS-programs can be run in background, reentrant or as RT (real-time) programs. Remember to use the corresponding library. The reader not familiar with these terms is referred to the relevant SINTRAN manuals.

## 3.2 *USE OF PICTURES*

PICTURE BUFFERS

Each application program using the screen handling facilities must define three data areas of fixed format:

1. the terminal array. This is normally named ITERM and contains information about the type of VDU being used.

2. the private picture buffer. This is normally named IPRIV and contains the object pictures currently being used by a specific application program. The terminal array and private picture buffer are contained in a Fortran labelled common area called PRIVATE.

3. the public picture buffer. This is normally named IPUBL and contains current object pictures. It is contained in a Fortran labelled common area called PUBLIC. It has the same layout as the private buffer, but can be accessed by several programs. Setting a flag in the ITERM data area enables the program to switch between the two picture buffers. See Fig. 3.1.

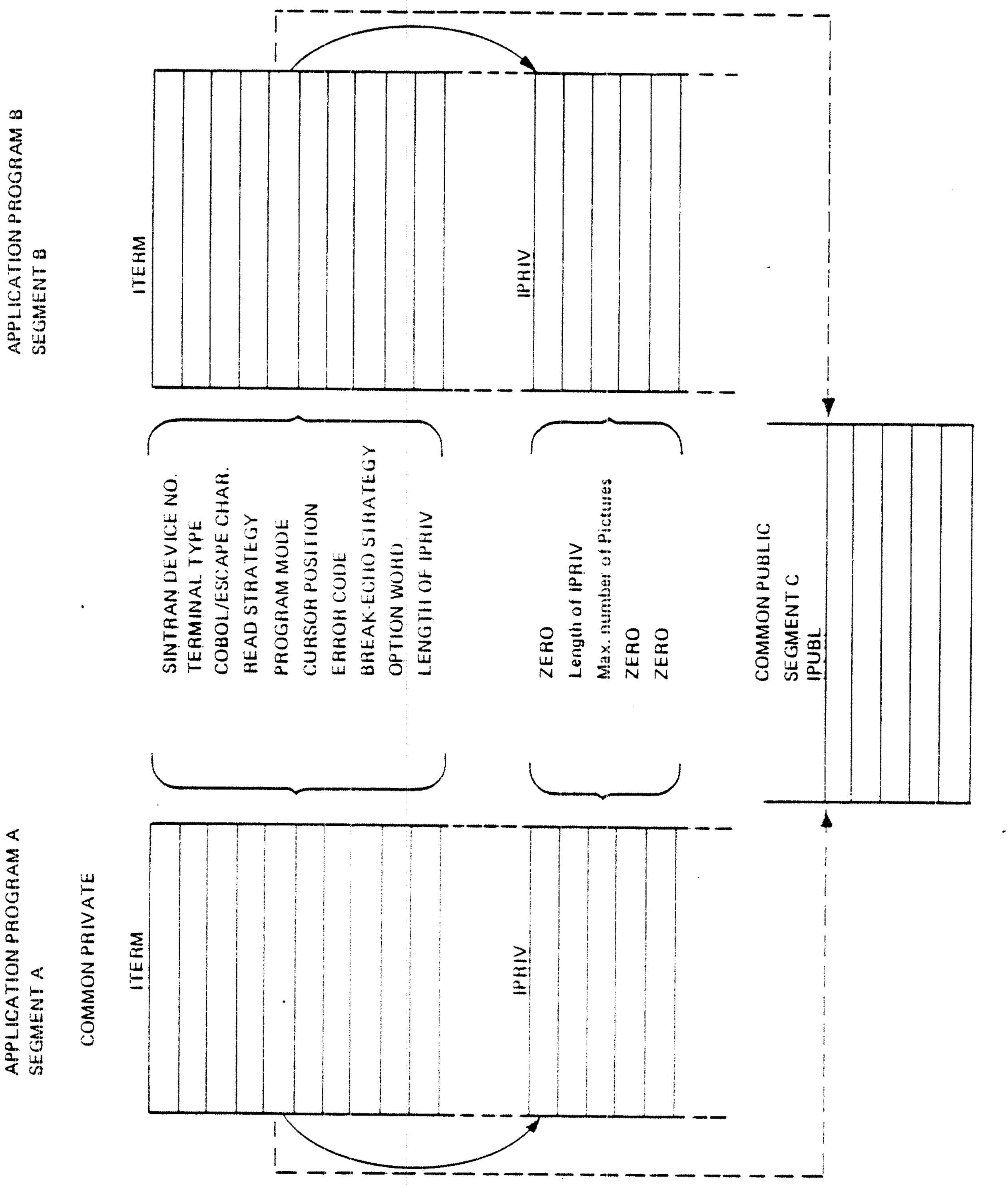


*Figure 3.1: Picture Buffers Used in NSHS.*

By a subroutine call (GTPIC) the application program can request specific pictures from the object file to be loaded into the picture buffer. See Fig. 3.2.

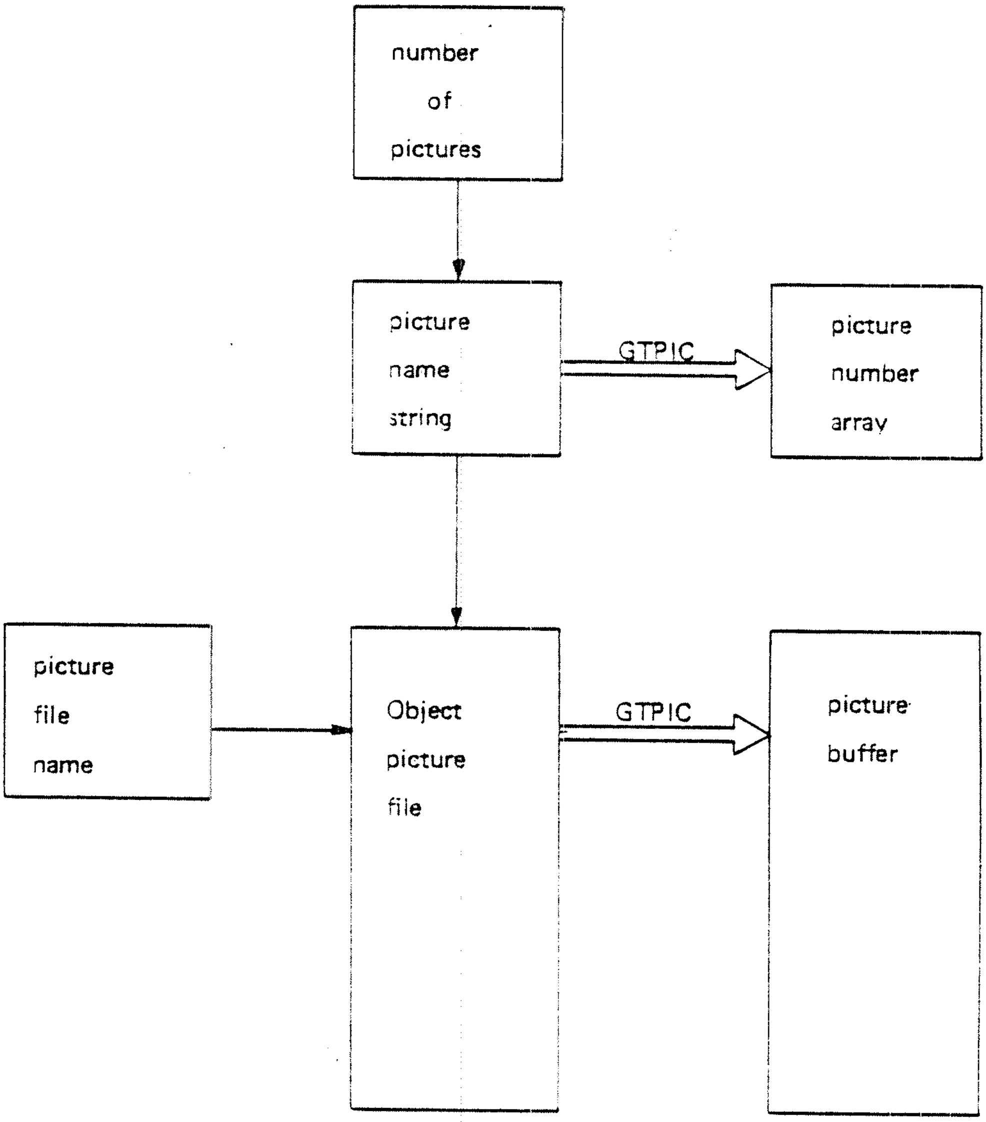


*Figure 3.2: GTPIC Subroutine - loads picture buffer and picture number array*

## FIELD ACCESS

All fields in a picture are given absolute field indexes in ascending order starting from line 1, position 1 and ending at the last position of the last line.

Before an application program can access a field it must create an array of field indexes or indicators, called the Field Indicator Array. This is passed to a NSL subroutine which converts the absolute indicators to field numbers which are placed in the Field Number Array; it is in this form that the field numbers are stored in the application program. See Fig.3.3.

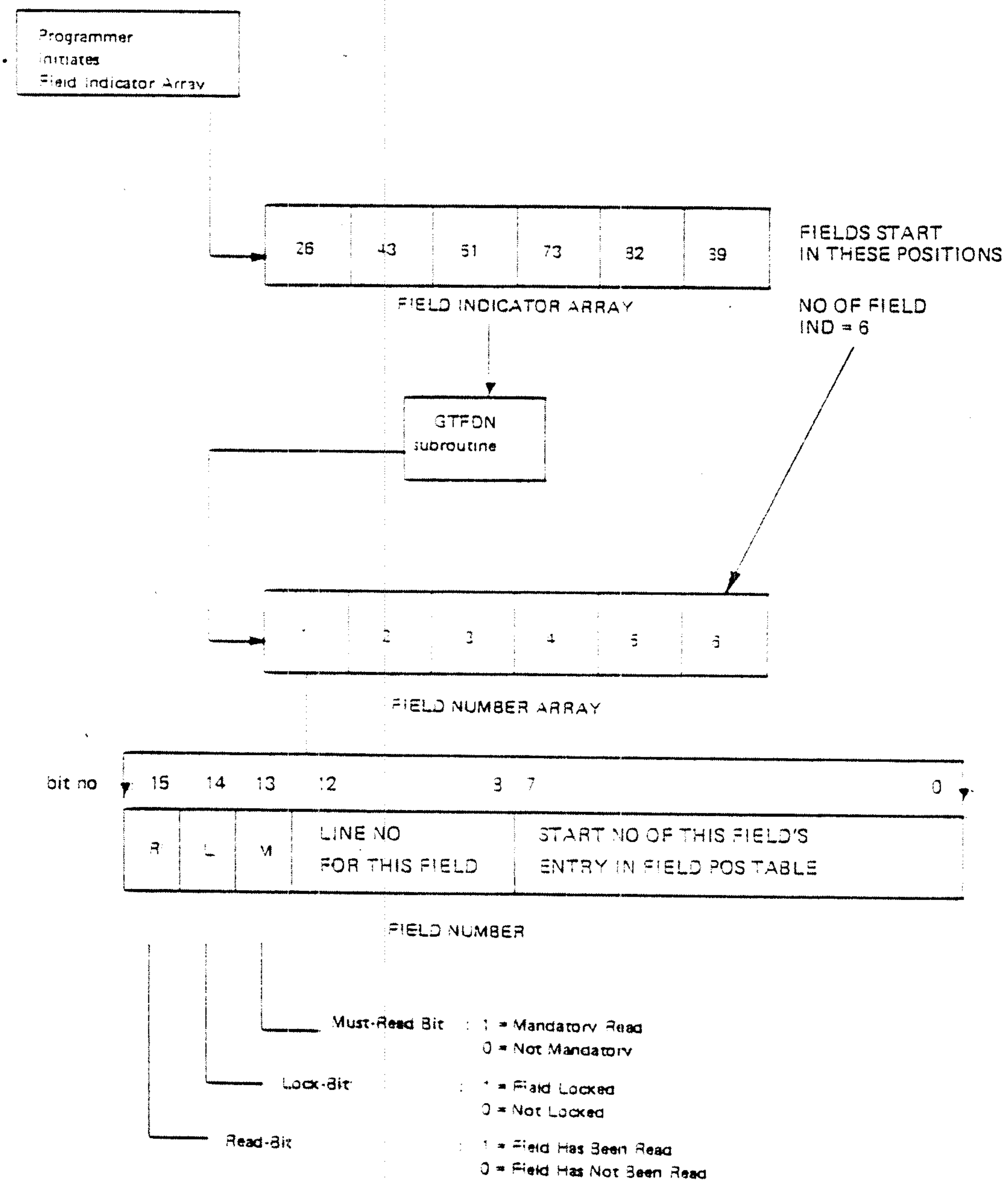


*Figure 3.3: GTFDN Subroutine - transforms absolute screen positions to field numbers*

The Field Number Array may then be used as a parameter in calls for input or output of data to or from the terminal. (See Figure 3.4.) The Field Number Array need not specify all fields in the picture. When reading data from the terminal, the application program can specify how many fields are to be read and at which field reading is to begin.

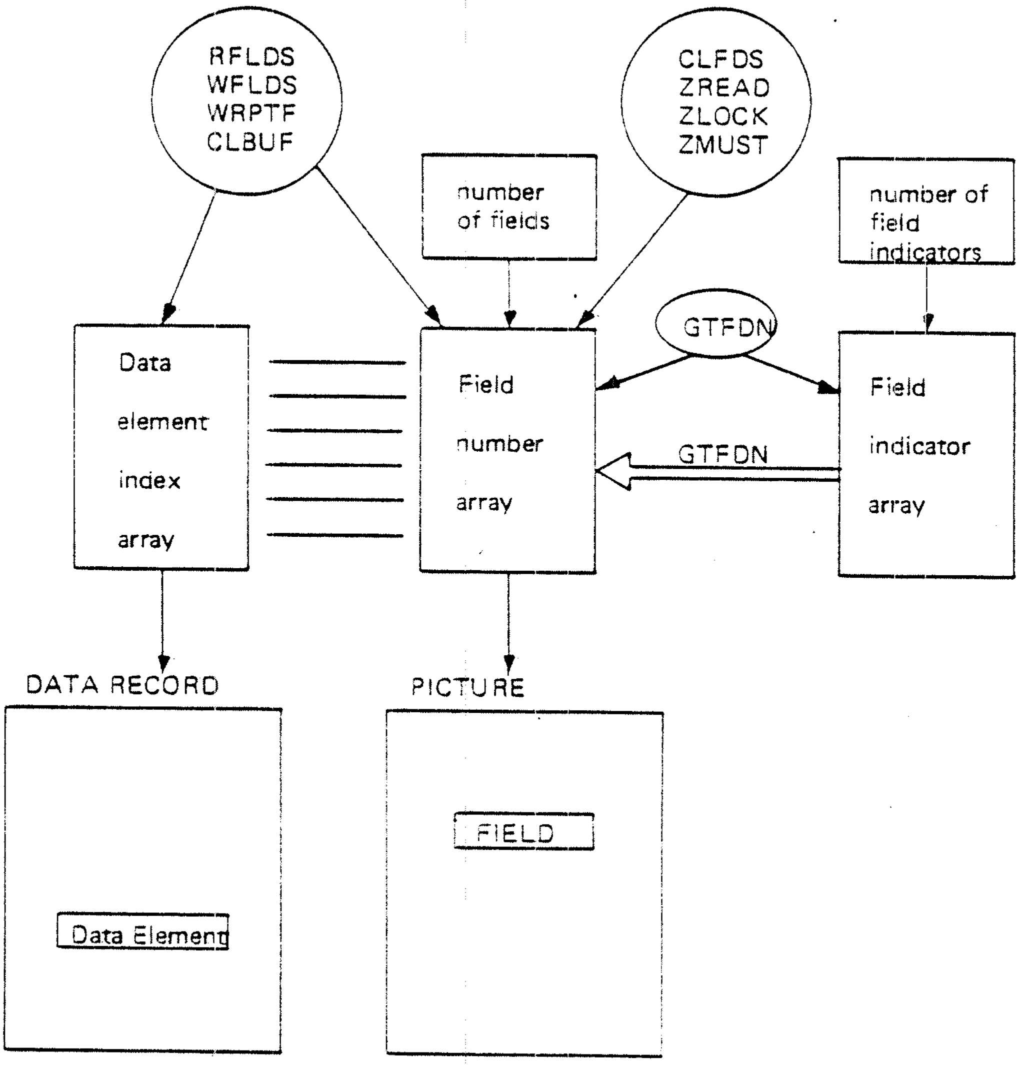


*Figure 3.4: References to tables in NSHS calls*

By creating several Field Number Arrays for the same picture, the application program can use the same picture in a varietv of ways. The same result is obtained by using a single field number array together with the locking/unlocking facility at runtime.

## DATA ACCESS

In NSL each picture corresponds to a data record, which is an integer array. Each field in the picture corresponds to a data element in the record. A Data Element Index Array is used when writing to or reading from the record. This array can be created by the application program so that its entries correspond directly to the entries in the Field Number Array, or if the first data element entry is given a value of zero NSL generates a default Data Element Index Array (see Figure 3.5).

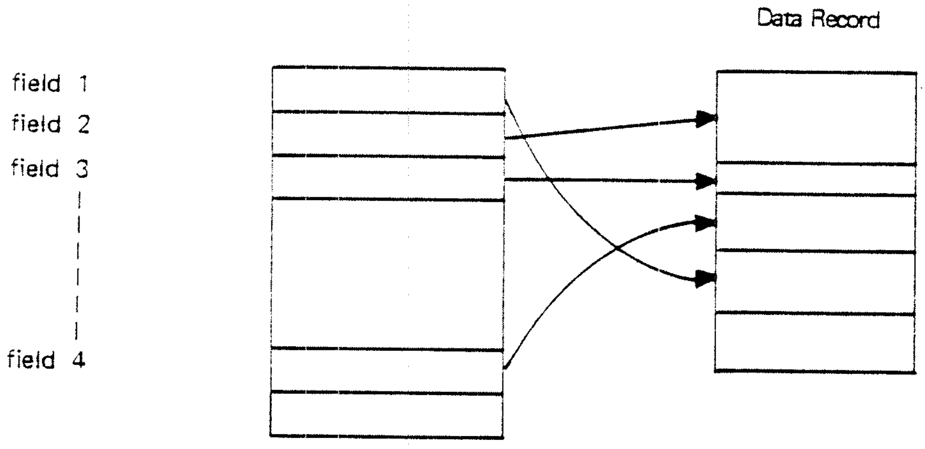


*Figure 3.5: Data Element Index Array*

In the Fortran program, or the data block written for Cobol programs, the private and public buffers are defined as follows:

```
    COMMON/PRIVATE/ITERM (128), IPRIV (1920)
    COMMON/PUBLIC/NROOTSEG, IPUBL (2047)
```

The lengths 1920 and 2047 are example values only. IPRIV and IPUBL lengths should be chosen so that the labelled common blocks are exact multiples of 1024. IPRIV must be defined to at least 30 words even if it is not used. Similarly, IPUBL must be defined to at least 5 words. For Cobol applications a Fortran datablock must be defined.

Common areas must start at the same location for all programs or segments. The private area can vary from a minimum of 128+30 words (no private pictures) to whatever size is required. The public area is normally the same size for all applications, as it is on a shared segment.

## 3.3 THE INITIAL CONTENTS OF THE COMMON AREAS

In the ND computer system a *word* is a datafield consisting of 16 bits numbered from 0 to 15. A *bit* is the basic element and has the value 0 or 1; this is *binary* notation. A bit with the value 1 is described as set, a bit with the value 0 is described as not set. Three bits can be grouped together and represented by an octal digit, which has a value between 0 and 7.

The first 128 words of the private area are called the terminal buffer and must be initiated correctly. The contents are as follows:

| Word | Contents |
|---|---|
| 1 | SINTRAN device number for this terminal |
| 2 | Terminal type and picture displacement |
| 3 | Cobol flag/escape character |
| 4 | Read strategy |
| 5 | PAD character/program mode |
| 6 | Cursor position |
| 7 | Error code |
| 8 | Break and echo strategy |
| 9 | Option word |
| 10 | Length of private picture area |

The remaining 118 words are used as a scratch area by NSL.

The length of the private picture area (word 10) should be set to 0 if no private pictures are to be used; IPRIV itself must be at least 30 words long.

A word sometimes only contains one piece of information. Eg. ITERM (4) contains read strategy. In other cases a word contains more than one piece of information. Eg. ITERM (2) contains terminal type and picture displacement. In order to set the bits in a word, the word is given a decimal or octal value equal to the binary value of the word. When the system encounters this octal or decimal value it unpacks it and sets the appropriate bits so that the word conveys the desired information.

More details of the datafields mentioned above are given in the following sections.

### 3.3.1 *ITERM(1) = SINTRAN Device Number*

The SINTRAN logical device number is given in Appendix C of the SINTRAN Reference Manual, ND-60.128. For application programs in background (programs started by a public user) the device number should be 1.

## 3.3.2 *ITERM(2) = Terminal Type/Picture Displacement*

Information about the type of terminal is necessary to ensure the particular terminal handles NSHS in the correct way. If the terminal type is known to SINTRAN the bit can be set automatically by call to a subroutine.

NSL maintains tables for each terminal type.

| 15 | 14-8 | 7-3 | 2-0 |
|---|---|---|---|
| Roll | Character displacement | Line displacement | Terminal type |

Picture displacement consists of character displacement (bits 8-14) which refers to the column number where the picture starts, and line displacement (bits 3-7) which refers to the line number where the picture starts. By giving this information the user can place the picture anywhere on the screen. This is of most importance when picture size is limited by the command DEF-PIC-SIZE and there are several pictures on the screen simultaneously.

Bit 15 is reserved by NSHS for setting page or roll mode operation.

The User Guide section has examples of how bits are set and how the value of a word is calculated.

## 3.3.3 *ITERM(3) = Cobol Flag/Escape Character*

Pressing the escape key interrupts execution of a read-fields call and returns to the main program. Control/G is frequently used for escape with the value 7.

For a Cobol program bit 15 equals 1. Some details of processing are different for Cobol programs.

## 3.3.4 *ITERM(4) = Read Strategy*

| | |
|---|---|
| 0 | Fields must be individually terminated. Return is given by typing control/S or one of the letters T,U,V,W,X,Y). |
| 1 | Fields must be individually terminated. Automatic return after reading or bypassing the last field, or leaving the first field using ↑ or ←. |
| 2 | Fields automatically terminated. Return as 0. |
| 3 | Fields automatically terminated. Return automatic, as 1. |
| 4-7 | As 0-3, but reedited fields are written out with blink. |
| 8-11 | As 0-3, but reedited fields are written out underlined. |
| 12-15 | As 0-3, but reedited fields are written out in low intensity. |
| 16-19 | As 0-3, but reedited fields are written out in inverse video. |
| 20-23 | As 0-3, but reedited fields are written out invisible. |

The last two groups refer to TDV 2100 terminals.

## 3.3.5 *ITERM(5) = PAD Character/Program Mode*

The right byte of this word contains the Program Mode character; it is 0 for private pictures, 1 for public pictures.

The left byte of the word contains the PAD character, used to fill the unused byte in a data element of storage type byte. This can be set by the user.

## 3.3.6 *ITERM(6) = Cursor Position*

This datafield is used internally by NSHS. It keeps track of the position of the cursor. It is initiated and updated by the system. Line number is found in the left byte, column number in the right byte.

## 3.3.7 *ITERM(7) = Error Code*

This is set to zero by the system when a subroutine is called; if an error is detected, an appropriate error code is inserted. See Appendix C.

## 3.3.8 *ITERM(8) = Break and Echo Strategy*

This decides how and when characters typed in on a RFLDS call are to be handled and echoed.

There are two strategies:

0 break on every character and immediate echo

1 the special break and echo strategy designed for NSL, which uses a feature in the SINTRAN III teletype routine. When a break occurs, or when a control or incorrect character is typed, the driver echoes several characters which are legal for the field (up to the maximum allowed for the field).

More detailed information is given in Appendix I.

## 3.3.9 *ITERM(9) = Status-Option Word*

This is used as a global flag in the NSHS system. Each bit has a special meaning.

Status/option-word has frequently been changed with new versions of the NSHS-system. The user should check the "Program Description" sheets for the actual version of the system. For most applications the option-word can be set to zero.

A more complete description is given in Appendix G.

## 3.3.10 *ITERM(10) = Length of Private Picture Area*

This value should correspond to the length defined for IPRIV in the Common block. In the example in Section 3.2, 1920 words have been allocated to IPRIV and in this case ITERM(10) = 1920.

## 3.3.11 *ITERM (11—128)*

These elements must be set to zero before the first NSHS-subroutine is called.

## 3.4 *THE PRIVATE PICTURE AREA*

The array IPRIV is initiated as follows:

| Word | Contents |
|---|---|
| 1 | 0 - zero |
| 2 | length of IPRIV; this must be equal to ITERM(10) |
| 3 | maximum number of picture descriptions which are to be read to IPRIV |
| 4 | 0 - zero (used for number of pictures already in IPRIV) |
| 5 | 0 - zero (used for number of words still available for new pictures) |

## 3.5 *PUBLIC PICTURES*

The PUBLIC picture array (IPUBL) can be initiated and used in the same way as the PRIVATE picture area. It is designed to be on a reentrant, or read-only segment and is normally preloaded.

The use of public pictures requires more knowledge of SINTRAN and the RT-Loader than could be expected of most public users. The use of reentrant and RT applications is not described in this manual, but an overviuw is given in Appendix J.

The arrangement most suitable for a particular application can be chosen. Eg. many terminals, few pictures, use the public area; many terminals, many pictures, a few used often, put commonly used pictures in the public area and read others to the private area when needed.

## 3.6 PROGRAM CALLS AND PARAMETERS

### 3.6.1 General

This chapter describes the parameters and call statements which activate the Fortran subroutines used in NSHS programming. Some subroutines have different entry points and parameters when they are called from a Cobol program.

### 3.6.2 Available Calls

GTPIC: get pictures; reads a number of pictures into the picture buffer and translates picture names to picture numbers which are then used to refer to pictures.
GTPICC: Cobol entry point
GTFDN: get field numbers; translates a list of field indexes/indicators to field numbers which are then used to refer to the fields.
RFLDS: read fields; reads a number of fields.

CLSCR: clear screen; clears all or part of the VDU screen.
CFLDS: clear fields; clears a number of fields on the VDU, writing full stops for each possible character position.
CLBUF: clear buffer; all nonlocked byte fields are filled with spaces, others with zero (binary).

WRPTD: write picture to display; writes a picture to the VDU. All I/O fields are blanked out.
WFLDS: write fields; writes fields to the VDU.
WRPTF: write picture to file; writes a picture with leading texts and fields to a file.

WMSGE: write message; writes a given text on the last line of the screen.
WMSGEC: Cobol entry point.
CLMSG: clear message; clears a message from the screen.

ZREAD: set read; sets bit 15 of the given field numbers to 1.
RREAD: remove read; sets bit 15 of the given field numbers to 0.
ZLOCK: set lock; sets bit 14 of the given field numbers to 1.
RLOCK: remove lock; sets bit 14 of the given field numbers to 0.
ZMUST: set must read; sets bit 13 of the given field numbers to 1.
RMUST: remove must read; sets bit 13 of the given field numbers to 0.

ZVERI: set verify on all fields described; the picture sets the lock bit on all fields which are not to be verified.

RVERI: reset verify.

ZMALL: set must all; sets must on all fields described.

RMPIC: remove picture(s); removes one or more picture descriptions from the private picture area.

ZBELL: rings the bell on the terminal.

The parameters and subroutines are explained in detail in the following sections.

## 3.6.3 *THE PARAMETERS IN NSHS CALLS*

The parameters are single integers, integer arrays or character strings and are defined accordingly in the application program. The way parameters are defined is different in Fortran and Cobol programs.

### 3.6.3.1 FORTRAN PROGRAMS

Single integers are automatically assigned if the first letter of the parameter name is I (or J,K,L,M,N). Integer arrays are defined in the program head and do not need prefix letters. Character strings are defined in the program head with a number of bytes.

An integer takes one computer word (16 bits) and one character is equivalent to one byte. Two bytes make one word. Character strings should be defined with an even number of bytes.

Data values can be preset in the heading by data statements (background programs) or in the body of the program by replacement statements (reentrant or RT programs).

Parameter names can be used in the subroutines, or the actual value can be set. Parameter names are often abbreviated to a few characters. A name may contain 7 significant characters.

### 3.6.3.2 COBOL PROGRAMS

All parameters must be defined in the working storage section of the application program. Integers are identified by the COMPUTATIONAL (or COMP) clause. All other parameters are defined by the PIC clause on the appropriate level.

In Cobol there is no restriction on the length of parameter name and no need for prefix letters. The parameter name must not coincide with any of the Cobol reserved words.

In the subroutine calls only formal names can be used and the values must be set by the VALUE clause in WORKING-STORAGE-SECTION (background programs) or by assignment in the body of the program (reentrant or RT programs). Integer constants are preferably given a name corresponding to the value they hold eg. W-1 holds the value 1, W-2 holds the value 2, etc.

### 3.6.3.3 STANDARD PARAMETER NAMES

Use of 'standard' names for parameters makes programs easier to read and errors easier to see. A list of 'standard' names used in this manual is given at the end of this section.

### 3.6.3.4 PICTURE-FILE-NAME

This is given in the normal way with name and type. It must be specified as a character string when input from Fortran. From Cobol it must be an alphanumeric variable in DISPLAY format defined on the 01 or 77 levels.

The file must be contiguous and preferably type :OBJ.

### 3.6.3.5 NUMBER-OF-PICTURES

The maximum number of pictures that can be called from an application program is 8 so this number will be in the range 1 to 8. The picture names are defined in the picture-name-string.

### 3.6.3.6 PICTURE-NAME-STRING

This string specifies the names of pictures to be used. Each picture name must fill 8 characters; names containing less than 8 letters must be padded out with spaces. It must be specified as a character string when input from Fortran. From Cobol it must be an alphanumeric variable in DISPLAY format defined on the 01 or 77 levels.

### 3.6.3.7 PICTURE-NUMBER-ARRAY

This is an integer array containing the picture numbers provided by the system.

Each picture number identifies a particular picture.

The number of words defined for the array should be at least twice the number of pictures used in the application.

### 3.6.3.8 PICTURE-NUMBERS

Each picture number refers to a particular picture. The numbers are generated by the call GTPIC and stored in the picture-number-array. The order is given by the picture-name-string; the first picture is 1 etc.

### 3.6.3.9 FIELD INDICATOR ARRAY

This array defines the fields within a picture. The fields in the array are only those that will be used.

Field indicators can be:

| | |
|---|---|
| Line number * 256 | all the fields on a line |
| Line number * 256 + position | a single field on the specified line |
| 1 - 255 | absolute field number |
| 0 | all the fields in a picture |

The field indicators in an array must be given so that fields are referred to in ascending order of field number.

The number of words reserved for this array should be equal to the maximum number of fields in one picture. Fields that have been defined as identical are counted as different fields.

### 3.6.3.10 FIELD NUMBERS

These numbers are translations of the field indicator array for a particular picture. They are supplied by the system and used as identifiers when fields are being read or written to. Field numbers contain information as to whether the field is locked, whether it has been read after a read command and whether input to the field is mandatory. A field number fills 1 word.

### 3.6.3.11 FIELD NUMBER ARRAY

This array of field numbers is generated from a field indicator array.

### 3.6.3.12 NUMBER OF FIELDS

This is an integer variable indicating the number of fields in the field number array.

## 3.6.3.13 RECORD

This is a data buffer corresponding to the fields on the screen. It receives data from the screen on reading or transmits data to the screen on writing.

The record is defined in the application program as an integer array. Its size should be the sum of the sizes of each datafield.

## 3.6.3.14 DATA ELEMENT INDEX ARRAY

This array contains the indexes to the first word of each data element in the record array. The indexes must be in one to one correspondance with the field numbers in the field number array.

If the first index in a data element index array is zero, the system automatically generates a data element index array with data items placed in the record in an order corresponding to the field number array. Indexes must be positive integers, greater than zero and less than or equal to the value in INDMX which is set to 2048; this can be changed by patching if necessary.

Output to the record can be edited in any desired order by means of the data element index array. See Section 3.2 and Figure 3.5.

## 3.6.3.15 STATUS

This is the last parameter in all calls and indicates if the call was carried out successfully or not.

| | | |
|---|---|---|
| STATUS = | 0 | no errors detected |
| = | N | where N is a positive or negative integer, error detected. |

In the case of an error a more specific error indication is placed in the datafield ITERM(7). All calls should be checked immediately on Status, the contents of Status and ITERM(7) written out and the program terminated.
In the case of RFLDS the Status can also be used for input (see section 3.6.4.3).
See Appendix C for further information.

## 3.6.3.16 STANDARD PARAMETER NAMES

| Formal name | Data type | Fortran name | Cobol name |
|---|---|---|---|
| *PICTURE-FILE-NAME | character string | PFNS | PIC-FI-NAM |
| *NUMBER-OF-PICTURES | integer | NOP | NO-OF-PIC |
| *PICTURE-NAME-STRING | character string | PNAS | PIC-NAM-STR |
| *PICTURE-NUMBER-ARRAY | integer array | PNA | PIC-NO-AR |
| *PICTURE-NUMBER | integer | IPN | PIC-NO |
| NUMBER-OF-FIELD-INDICATORS | integer | NFI | NO-FI-IND |
| *FIELD-INDICATOR-ARRAY | integer array | FIA | FI-IND-AR |
| FIELD-INDICATOR | integer | IFIND | FI-IND |
| *FIELD-NUMBER-ARRAY | integer array | FNA | FI-NO-AR |
| *FIELD-NUMBER | integer | FNO | FI-NO |
| *NUMBER-OF-FIELDS | integer | NOF | NO-OF-FI |
| *RECORD | integer array | REC | SCR-REC |
| *DATA-ELEMENT-INDEX-ARRAY | integer array | DEIA | DAT-EL-IX-AR |
| NUMBER-OF-FIELDS-READ | integer | NOFR | NO-FI-RE |
| TERMINATING-CHARACTER | integer | ITERCHA<br>or ITERM | TER-CHA |
| WRITE-CODE | integer | IWCOD | W-CODE |
| READ-CODE | integer | IRCOD | R-CODE |
| CLEAR-CODE | integer | ICLCOD | CL-CODE |
| FILE-NUMBER | integer | IFI | FI-NO |
| USER-CONTROL-NUMBER | integer | IUSCOD | — |
| FLAG | integer | IFLAG | FLAG |
| READ-NUMBER-OF-FIELDS | integer | RNOF | READ-NO |
| LOCK-NUMBER-OF-FIELDS | integer | LNOF | LOCK-NO |
| MUST-NUMBER-OF-FIELDS | integer | MNOF | MUST-NO |
| START-INDEX | integer | ISTIX | ST-IX |
| INDEX-TO-FIELD | integer | INDEX | — |
| FIRST-LINE | integer | IFL | FI-LI |
| LAST-LINE | integer | ILL | LA-LI |
| START-OR-END-POSITION | integer | ISEP | ST-EN-PO |
| FIELD-INFORMATION-ARRAY | integer array | FINFO | — |
| TEXT | character string | TXT | SCR-TEXT |
| *STATUS | integer | IST | STATUS |

Parameters marked with * occur in more than one call and are explained in section 3.6.3. Parameters which are only used in one call are explained in connection with that call in section 3.6.4.

A few of the parameters are used in Fortran subroutines and have no corresponding Cobol names.

## 3.6.4 *The Subroutine Calls*

Parameters in italics show where the result of the operation is stored. Parameters occuring in more than one call are explained in section 3.6.3, others in connection with the call where they occur.

### 3.6.4.1 GTPIC (Get picture). Cobol call name: GTPICC

CALL GTPIC (picture-file-name, number-of-pictures, picture-name-string, *picture-number-array, status*)

This subroutine reads pictures into the current picture buffer addressed by ITERM(5) and provides picture numbers which can then be used to refer to the pictures in other subroutine calls. Nothing is written on the display.

This is usually the first NSHS-call in an application program.

### 3.6.4.2 GTFDN (Get Field Numbers)

CALL GTFDN (picture-number, number-of-field-indicators, field-indicator-array, *field-number-array, number-of-fields, status*)

This call translates a field indicator array into a field number array which is then used to refer to fields in other subroutine calls.

Field indicators within the array may be:

| | |
|---|---|
| 0 | all fields in a picture; number-of-field indicators must then be 1 |
| line number * 256 | all fields on a line |
| line number * 256 + position | an individual field on the specified line |
| 1 - 255 | absolute field number ie. an individual field. |

GTFDN checks that given field indicators are acceptable and then generates the corresponding field number array.

This is usually the second NSHS-call in an application program.

### 3.6.4.3 RFLDS (Read Fields)

CALL RFLDS (read-code, picture-number, number-of-fields, *field-number-array, record, data-element-index-array, number-of-fields-read, terminating-character, status*)

See also 4.5 about editing and Appendix G about Status/options.

The main purpose of this subroutine call is to read fields from the keyboard to the VDU screen and to convert the characters read to their respective data element value which is then placed in the record. RFLDS has five functions, indicated by the value of the input parameter read-code.

| *read-code* | *meaning* |
|---|---|
| 0 | normal read |
| 1 | control read (verify) |
| 2 | check record |
| 3 | read password(s) |
| 4 | change record |

In addition to controlling input to an individual field RFLDS also:

- determines whether a value read in is acceptable according to the defined control functions
- executes accumulation and equivalent control functions and writes the new values to the appropriate field
- manages tabbing functions
- provides comprehensive editing functions.
- ensures that fields with the read-must bit set cannot be bypassed without receiving a value
- ensures that locked fields are bypassed.

When a field is to be read, RFLDS places the cursor in the first position of the field on the screen. If the characters typed are legal for the field type they appear on the screen. The terminal gives an audible signal if illegal characters are typed or if you attempt to type more than the field has space for. (See section 3.3.4, Read Strategy.) When reading a password, no characters appear on the screen and no indication of illegal characters is given.

For all calls the last parameter Status is a receiving field. For RFLDS however, Status can also be used to indicate at which field reading is to start. Eg. if Status is set = 4, the cursor will be positioned at the beginning of field 4.

Input to a field is terminated automatically when the last character is accepted (depending on the read strategy) or explicitly by typing a terminating character (CR or a user defined character). After termination, characters read in can be rewritten to the field in their reedited form and/or with a display mode other than normal, depending on the terminal type.

In NSHS the use of 'writing' and 'reading' may be confusing for the operator. The transactions are seen from the record buffer:

> new values received by the record from the terminal or program means *reading*
>
> values already in the buffer transmitted to the screen means *writing.*

TERMINATING CHARACTER

The terminating character indicates how the read call is terminated and has one of the following values:

| | |
|---|---|
| —1 | user defined escape character |
| 0 | CR (terminating last field) |
| 1 | user defined field terminating character (terminating last field) |
| 7-14 | control/L, or control/S-Y |
| >99 | terminated by a user control function |

For read strategy = 1, see (3,3,4), terminating a field with the arrow keys will give the following values for the terminating character:

| Arrow | Value |
|---|---|
| <--- | 2 |
| ---> | 3 |
| ↓ | 4 |
| ↑ | 5 |

Cursor control characters and control/L or S-Y only terminate reading when the cursor is outside a field. If one of these control characters collides with a cursor control character (control/Y or Z on VISTA,· control/X on TDV 2100) this invalidates their function as a terminating character.

Depending on the current break strategy, control/V may be a special character in SINTRAN. This means that in practice the following can be used as terminating characters for a RFLDS call:

| | |
|---|---|
| TDV 2000 | control/L,S,T,U,W,X,Y |
| VISTA | control/L,S,T,U,W |
| TDV 2100 | control/L,S,T,U,W,Y |

**NORMAL READ (Read-code = 0)**

*Copy Field*

If you wish to correct a record which has been read in previously, it is written to the screen using WRPTD and WFLDS, or WRPTF. Fields which will not be changed are locked, those which will be edited are set read. The Read-bit in the appropriate field numbers is set to 1 before RFLDS is called; when the field is read old value characters are made available for editing/copying. Characters can be copied from identical fields immediately above the one being read (see section 5.2).

When reading in a new record you can tab back to a field already typed in and edit/copy its old characters.

*Setting the Read-bit (to 1)*

This occurs when a field is given a value. Read bits set on entry to RFLDS (to indicate old values) will be set to zero on exit from RFLDS if no value has been read into the field.

*Empty Data-Elements/Fields*

For numeric fields with byte storage code, an empty data element (containing all spaces) is different from a data element containing one zero + spaces.

An empty value can be forced into a field/data element during RFLDS by typing control/Q followed by control/J. The corresponding data element receives the empty value via the clear-buffer function and is then written out. For byte storage codes the data element is filled with spaces; for non-byte storage codes the result is the same as typing in a zero.

Typing control/Q, control/J gives the field a value and in a normal RFLDS call will be set-read. In a control-read it will be checked against the old data element value and if different but to be accepted, it will also be set-read.

*Control*

If user control is defined for a field, the control may be carried out before or after the field has been read depending on how the status in the control function is set.

The example on user control in the User Guide section demonstrates how a field can be accessed before and after reading.

*Number-of Fields-Read*

This is the number of fields to which characters have been typed and a new value accepted. Typing the same value again is equivalent to a new value.

CONTROL READ (Read-code = 1)

With this code, values read in are compared with values already in the record; comparison occurs as characters are typed and at the termination of individual field reading.

If a typed character does not match the old character string an audible warning is given and the cursor will not move. The next character is accepted provided it is legal for the field. If the character is a digit in a numeric field but still different from the original an audible signal is given and the old value displayed on the last line of the screen. A warning is given for subsequent characters which do not match, but future characters are accepted.

If values are different at field termination an audible warning is given and the cursor repositioned at the beginning of the field. The old value is displayed on the screen line. The new value is accepted if → is pressed, old value if ↑ is pressed, or the field can be retyped. ↓ gives a check if rest of the record is empty. The cursor stops on the first not-empty field if it exists, else RFLDS terminates.

*Editing*

The editing characters control/A or ← are legal. Control/Q resets everything as it was when typing began.

*Number of Fields Read*

For a RFLDS call with code 1 this is equal to the number of fields which have recieved a new value. On entry all fields not locked automatically receive old values and Must-read is set for all fields.

*Control Read (Verify) versus Normal Read (Original)*

Record verifying operations at the keyboard are identical to those used when entering a record via a normal RFLDS call. Eg.:

| Normal Read | Control Read |
|---|---|
| bypass using → | bypass using →<br>illegal if value in old field;<br>→ has same function as before if different value given. |
| control/Q,J | control/Q,J giving empty field.<br>Accepted if old field was empty; in this case → has the same function. Otherwise, the field is blanked out and the old value written on the last line. |
| CR | always accepted as zero value if old value was 0 (ie. accepted as if 0 + CR had been typed). |
| LF | bypass to first field on next line. |

| | |
|---|---|
| CURSOR DOWN | terminates a field and checks if the rest of the record is empty. The cursor stops on the first not-empty field if this exists, else RFLDS terminates. |

If old and new values are different after termination and the old value was an empty byte field the message "field was empty" is written on the last line.

*Termination of Control Read, Verify*

Using the verify function a predetermined order from top to bottom and left to right of the picture must be followed.

Control read can be terminated by control/Q twice followed by control/S; but there is no way of knowing which fields after the last one corrected have been verified.

In Data Entry, like all other commands Verify can be terminated by control/G (the escape character), but the current record is then unaffected.

CHECK RECORD (Read-code = 2)

Using this code, control functions defined for a record can be used on other records with the same format but which were originated independently of a normal RFLDS call. It does not affect the VDU screen.

Check record is executed when read-fields is called. On return, the read bit is set to 1 in all correct unlocked fields (ie. they are negative). The must-read bit is set to 1 in all incorrect unlocked fields.

If control functions are not defined for an unlocked field, no status bit is set. Number-of-fields-read is the number of data items controlled and found correct. If any items are incorrect, status on return from read-fields is 2 and ITERM 7 is 40.

READ PASSWORD(S) (Read-code = 3)

RFLDS is executed normally but none of the characters typed appear on the screen. The cursor moves from field to field, but no audible warning is given for incorrect characters. If the field has had control functions defined, they are executed. Control of passwords in an application program should take place externally to read-fields.

CHANGE RECORD (Read-code = 4)

When values are read to datafields in a picture (and record), the read-bits are set (=1) for these fields. If the record is called back for changing and you forget to reset the read-bits, it will be possible to pass 'must' fields. Using change record avoids this problem as it resets the read-bits automatically.

For relation between storage-code, edit-code, fill-code and record contents, see 6.4.1 and Appendix A.

### 3.6.4.4 CLSCR (Clear Screen)

CALL CLSCR (clear-code, first-line, last-line, start-or-end-position, *status*)

This routine call clears all or part of the VDU screen.

Possible values for clear-code are:
0 clear whole screen
1 clear whole lines
2 clear from and including start position to end of lines
3 clear from and including position 1 to and including end position.

Lines to be deleted are indicated by giving the numbers of the first and last lines. If only part of the line is to be deleted the clear-code is set to 2 or 3; the start-or-end position accordingly clears the first or last part of the line.
If different parts of lines, or a number of single lines are to be deleted, the call must be repeated.

### 3.6.4.5 CFLDS (Clear Fields)

CALL CFLDS (picture-number, number-of-fields, field-number-array, *status*)

This subroutine call clears a given number of fields in a picture (ie. puts dots in all character positions). Locked fields are not cleared.

The data record is not affected.

### 3.6.4.6 CLBUF (Clear Buffer)

CALL CLBUF (picture-number, number-of-fields, field-number-array, record, data-element-index-array, *status*)

This subroutine call zeroes out data elements in the record. For storage code 3, the data element is filled with spaces, for other storage codes it is filled with binary zeroes. Read-bits for all non-locked fields are set to zero.

## 3.6.4.7 WRPTD (Write Picture to Display)

CALL WRPTD (picture-number, *status*)

This subroutine writes a picture on the display showing all leading texts but with all fields blanked out and dots in each character position.

Only one picture is written out per call, but it is possible to have any number of pictures visible on the screen at one time provided they do not overlap.

## 3.6.4.8 WFLDS (Write Fields to VDU)

CALL WFLDS (write-code, picture-number, number-of-fields, field-number-array, record, data-element-index-array, *status*)

This subroutine call writes a specified number of fields to the VDU. Write-code is an input parameter which selects the display mode. It can have the values:

| | |
|---|---|
| 0 | normal |
| 1 | blink |
| 2 | underline |
| 3 | low intensity |
| 4 | inverse video |
| 5 | invisible |
| 6-11 | as 1-5 but with bell |

These codes have no effect if the particular display mode is not available on the type of terminal being used.

Only *unlocked* fields are written if status on input has the normal value=0. If status ‹0 on input to WFLDS, only *locked* fields are written.
Single or double integer storage code data elements which are defined with sign suppress and have negative values, or have been given values containing more decimal digits than the field allows (overflow) will be written out with asterisks instead of digits.

Byte storage code data elements containing bytes with values less than 040 octal will be replaced by asterisks; inconsistent data elements for storage codes 4 and 5 will also be represented by asterisks.

### 3.6.4.9 WRPTF (Write Picture to File)

CALL WRPTF (file-number, flag, write-code, picture-number, number-of-fields, field-number-array, record, data-element-index-array, *status*)

This subroutine writes a picture with leading texts and fields (except locked fields) to a file. The file can be the display, so that WRPTF can be used as WRPTD with field values.

The parameter flag is used to distinguish between normal sequential media and the VDU screen and can have the values:

1 do not write display mode characters, use space and line feed.
0 write display mode characters.

The parameter write-code can have values 0 - 5 corresponding to the first 6 values of the same parameter in WFLDS. The file number must be recognised by an OPEN statement in the application program. If COBOL, use call to OPENFC, see Appendix H-and program examples.

### 3.6.4.10 WMSGE (Write Message) Cobol call name: WMSGEC

CALL WMSGE (Text)

This subroutine writes a message on the last line of the VDU screen.

Text must be a variable containing up to 79 characters terminated with an apostrophe or *. All non-printable characters will be replaced by *. Text must be a character string when called from Fortran and an alphanumeric variable in DISPLAY format defined at the 01 or 77 levels when called from Cobol.

### 3.6.4.11 CLMSG (Clear Message)

Clears message on screen and corresponds to WMSGE. No parameters.

### 3.6.4.12 ZREAD/RREAD, ZLOCK/RLOCK, ZMUST/RMUST

CALL ZXXXX or RXXXX (picture-number, number-of-fields, start-index, field-number-array, *status*)

These subroutine calls set (to 1) or reset (to 0) the three status bits in a field number. See fig. 3-3.

On entry to a RFLDS call, the read bit indicates whether or not there is a value for the field in the record (1 or 0) and on return from a RFLDS call it indicates whether or not a value has been read to the record (1 or 0). Even if the read bit is 1 on entry to RFLDS it will be 0 on return if no new value has been read in.

The lock bit is used to lock a field; it cannot then be written from or read into and is treated as a leading text. Locked fields will not appear if WRPTF is used.

The must bit is used to set a field so that once entered, the field must be given a value. CR only (giving zero value) is not accepted. A space or 0 must be typed. (This bit has meaning only for RFLDS.)

Start-index indicates which field number in the field number array you are starting at.

### 3.6.4.13 ZVERI/RVERI, ZMALL/RMALL

CALL ZXXXX or RXXXX (picture-number, number-of-fields, field-number-array, *status*)

These calls act on the relevant bits as described in Section 3.6.4.12. They work only for those fields defined with must or verify during picture definition. (See Section 2.4.)

### 3.6.4.14 RMPIC (Remove Picture)

CALL RMPIC (number-of-pictures, picture-number-array, *status*)

This subroutine call removes pictures from the private picture buffer. If the number of pictures is zero, the private buffer will be cleared.

### 3.6.4.15 ZBELL (Ring Bell)

```
CALL ZBELL
```

Gives an audible signal on the terminal. No parameters.

# 4 THE TERMINAL OPERATOR'S JOB — USING THE SCREEN PICTURES

## 4.1 *GENERAL*

NSHS provides highly automated data processing and reduces the number of errors by releasing the operator from complicated tasks. If the programs controlling the processing are well organized and error free, very few mistakes will occur.

NSHS has several different options, but the operator's job is usually to enter data to a picture on the terminal screen. The picture often resembles the actual source document. As data are entered at the keyboard the picture fields are filled.

If unacceptable data are entered, the terminal gives an audible signal and the data are rejected. An error message can be programmed to appear on the bottom line of the screen. The cursor is placed at the beginning of the field and you can continue with acceptable data. It is necessary to have a picture description at hand to show which data are acceptable in different fields.

During editing, the tools developed for screen handling are used. These are explained in Section 4.5. They will be learnt by practice — try them out for yourself on a terminal!

Fields may be designated must-fields, meaning that a value must be given to that field. For a numeric field this can be CR, giving the value 0.

Fields may be locked-fields, meaning that it is not possible to give a value and the field is jumped over by the cursor. Any attempt to violate these rules produces an audible signal from the terminal.

The cursor always indicates where you are on the screen and is moved by the arrows on the keyboard.

When all fields in the picture are completed the fields are cleared and the cursor repositioned at the start of the first field. The data are transfered to a file or database according to the application program.

## 4.2 *GETTING STARTED*

Assume you are going to enter data for a person register. The job is called REGISTER-PERSONS, or abbreviated to REG-PER. The steps below explain what to do, starting with switching on the terminal and entering the system.

1. Switch on power to the terminal; press the line key. Some of the lights on the keyboard light up and the cursor appears at the top left of the screen (home position).

2. Press the escape key. After a few moments the following text appears:

   ENTER: j-b CR — Answer with your user name, abbreviated (which must be in the main directory) followed by CR.

   PASSWORD: CR — Give your password if you have one, otherwise just CR.

3. @REG-PER CR — At the SINTRAN sign write the name of the job and press CR.

The job is now started. Depending on the particular job you may get questions to answer or a picture directly on the screen.

## 4.3 *IF SOMETHING GOES WRONG*

1. Nothing happens when escape is pressed.
   You may be off-line. Press 'line' and 'escape' again.
   If you are off-line you can still write characters on the screen; if this is not possible the terminal is not connected or the computer is down.

2. The error message NO SUCH USER is given when you enter your user name.
   You may have misspelled your name. Try again. If the response is the same, contact the system supervisor.

3. ENTER appears again when you type in your password.
   The wrong password was given. Start from the beginning again. If you have no password press CR. If you have forgotten your password see the system supervisor.

4. You may be asked questions about PROJECT NO. etc. Answer with CR unless you have other instructions.

When @ appears you can give any SINTRAN command available to public users.
Program names with the type :PROG can be used like commands.
Commands must follow immediately after the @ sign. "Noise" on the line may move the cursor a few positions and any command given will then be rejected. Just repeat the command.

## 4.4 *TERMINATING THE JOB*

With the cursor in the first field press control/G, the usual escape command in NSHS. The SINTRAN sign appears. Proceed with a new job name and CR, or finish work at the terminal by typing LOG and pressing CR.

## 4.5 *EDITING*

Editing of fields follows the same strategy as picture definition, except that editing is oriented towards fields, not lines. Only identical fields lying directly above can be copied, using control/P or R and ↑. Editing characters are not copied.

The following control characters are used in editing:

| | |
|---|---|
| control/A | delete previous character |
| control/B X | copy identical field above up to and including X |
| control/C | copy one character from old value |
| control/D | copy old value to end of field without terminating the field |
| control/E | insert characters until the next control/E is typed |
| control/F | at the beginning of a field, write out version number |
| control/G | multipunch; hexadecimal code for character given on last line |
| control/N X | copy field above up to X |
| control/O X | copy old characters in this field up to but not including X |
| control/P | copy one character from identical field above |
| control/Q | delete all characters in this field |
| control/R | copy identical field above and terminate field; legal only with cursor in first position of field |
| control/Z X | copy old characters in this field up to and including X |
| ontrol/K X | as control/Z X (for Vista and Infoton 200 only) |
| control/Q<br>control/J | gives field a zero value |
| control/Q<br>control/A | blanks out other fields in the picture/record |
| line feed | go to first field on next line that has fields. |

When inside a field the cursor control characters can be used as follows:

| | |
|---|---|
| → | equivalent to control/C |
| ↓ | equivalent to control/D but terminates the field. In verify mode, check if the rest of the fields are empty. The cursor will stop in the first position in the first non-empty field or else terminate the record. |
| ← | equivalent to control/A |
| ↑ | equivalent to control/R |

Insert mode is initiated or terminated by control/E. It is not visibly indicated, but every character typed is echoed audibly. Removal of inserted characters using control/A or ← produces an audible signal even after leaving insert mode.

Multipunch enables the user to insert any character value (0-377B) in the data element for a string field. The value for the character is represented on the screen as a star. Multipunch characters with values greater than 337B or which represent characters illegal for the field type cannot be copied via control/C,D etc., they must be entered again.

NUMERIC FIELDS

If numeric fields have a sign position, the sign can be given as either the first or the last input character.

It is allowed to give leading zeroes.

Decimal fields (edit-code 0-9) can be terminated with either CR or SPACE. If field is terminated with CR, the decimal point will be placed after the last typed digit. The positions behind the decimal point is filled with zeroes. When space terminates the field, the last typed digit will be placed in the last significant position behind the decimal point. if the decimal point is typed, the field will be edited according to typing.

## 4.6 *AUTOMATIC PICTURE RECOVERY WITHIN READ-FIELDS*

The screen picture may disappear for various reasons eg. static electricity, noise, power failure at the terminal, accidental switching off etc.

The automatic recovery system clears the screen, writes the picture displaying filled and set read fields and positions the cursor at the start of the current field. It is called by pressing control/Q three times.

# 5 NSHS SUPERVISOR SECTION

## 5.1 *GENERAL*

The supervisor has responsibility for carrying out certain tasks for all users. One of these tasks is to implement new software such as NSHS.

Three kinds of users are defined in the ND computer system; PUBLIC, RT and SYSTEM. Users logged in under their own user name are PUBLIC users. To make software accessible to all users it is loaded into the SYSTEM user area, which is administered by the supervisor and protected from unauthorized use by a password.

The supervisor must have a considerable knowledge of SINTRAN. In addition to implementing NSHS, the supervisor prepares public pictures and loads segments on the seg-file. The following sections explain the supervisor tasks connected with NSHS.

## 5.2 *THE MATERIAL*

The NSHS system is delivered on three diskettes accompanied by loading instruction sheets. These sheets must be available to the supervisor as they contain information necessary for correct implementation. Details vary from one version to another — do not use loading instructions for an old version!

Diskette no. 1 contains the main modules of the NSHS system, the picture definition and libraries. No. 2 contains additional service programs, and no 3 the libraries for the 1- and 2 BANK-system. The diskettes are accompanied by file descriptions. Figure 5.2 shows a description for the current version and figure 5.3 shows a loading instruction sheet. Versions are indicated by the last letter of the filename (eg. SCREEN-DEF is version H). Files should be implemented with their full names; this makes it easier to check the version later. Modules are updated to add new features or remove errors; it is part of the supervisor's job to look for these new versions.

Diskette 1:

(ND-10013I -PART1:FLOPPY-USER)SCREEN-DEF-2155I:BPUN;1
(ND-10013I-PART1:FLOPPY-USER)SCREEN-LIB-2154K:BRF;1
(ND-10013I-PART1:FLOPPY-USER)SCREEN-RTL-2156K:BRF;1

Diskette 2:

(ND-10013I-PART2:FLOPPY-USER)SCREEN-COP-2446A:BPUN;1
(ND-10013I -PART2:FLOPPY-USER)SCREEN-UCO-2447B:SYMB;1
(ND-10013I-PART2:FLOPPY-USER)SCREEN-DEM-2181C:SYMB;1

Diskette 3:

(ND-10013I-PART1:FLOPPY-USER)SCREEN-1BANK-A:BRF
(ND-10013I-PART1:FLOPPY-USER)SCREEN-2BANK-A:BRF
(ND-10013I-PART1:FLOPPY-USER)SCREEN-1REEN-A:BRF

The file type gives important information about how the file is used. The table below shows examples of file types and how they are handled.

| File Type | Sintran Command | Use |
|---|---|---|
| :SYMB | @COPY-FILE | This is a source file which can be edited by the user for his own special purpose. It must be compiled before use and gives a :BRF-file and if dumped a :PROG-file. |
| :BRF | @COPY-FILE | This is a compiled program ready to be loaded by the user. Libraries are usually of this type. |
| :BPUN | @PLACE-BIN<br>@DUMP<br>or<br>@DUMP-REENT | Stand-alone programs for loading by the user. The first alternative gives a :PROG-file and can be found by the @LIST-FILE command. Dump-reentrant puts the program on a segment and is found by @LIST-REENT. In both cases the program is started by giving the program name as a command in SINTRAN. |
| :PROG | @COPY-FILE | Programs in executable format in the file system. They are started by giving the name like a command in SINTRAN. |

Table 5.1 File types on the diskettes and how they are handled.

## 5.3 *IMPLEMENTING NSHS ON THE COMPUTER*

If there is enough space for user SYSTEM on the main disk it is convenient to place the files from the diskett on the disk. Before the files can be copied the diskett must be entered so it is known to the system. The procedure for diskette 1 is given below; it is the same for diskette 2 and 3 except for the different directory name.

Place the diskett in the floppy slot.

| | |
|---|---|
| @ENTER-DIR CR | The SINTRAN command to enter the diskette. |
| DIRECTORY NAME: n-1-p CR | This abbreviation is sufficient. |
| DEVICE NAME: f-d-1 CR | The abbreviation for floppy-disk-1, a standard name. |
| UNIT NO: 0 CR | This is the leftmost unit of the floppy station. If there is more than one slot, they are numbered 0,1,2... from the left. |

There may be obstacles to this procedure, eg. the floppy station may already be occupied.
@LIST-DIR CR shows if this diskette is entered; if it is not entered it can be removed.

When the NSHS diskette is successfully entered
@LIST-FILES(n-1-p:f-u),,, CR displays all files on the diskette.

It is possible to copy all these files to the disk in one operation by means of the backup-system (@backup CR). However, copying one file at a time is described here.

| | |
|---|---|
| @COPY-FILE CR | The SINTRAN command for copying files. |
| DESTINATION<br>FILE:"screen-def-2155l:boun" CR | The file name betweem quotes because the file is being created; the name is given in full. |
| SOURCE-FILE:(n-1-p:f-u)s-def:bp CR | This name can be abbreviated provided this does not produce ambiguity. |

If no error message is given and @is displayed, proceed with the next file.

When all relevant files are copied, the :BPUN-files are dumped following the procedure described on the loading instruction sheet. The PLACE and DUMP commands mentioned in Table 5.1 are used:

```
@PLACE-BIN  scr-def:bpun  CR
@DUMP  "screen-def-2155I",0,0  CR
```

Parameters can be given immediately after the command. Start and restart addresses are 0 for both :BPUN-files in this case. :SYMB-files can be edited if desired and compiled to produce the :BRF-version. This is described in the relevant manuals.

# 6 USER GUIDE SECTION

## 6.1 *INTRODUCTION*

Use of NSHS involves the following stages:

1. Creating the picture files (contiguous).
2. Creating pictures with the desired field control and eventually user control.
3. Writing application programs for different tasks, using the pictures.
4. Writing User Control programs if specified in any picture.

The practical procedure is slightly different for Fortran and Cobol programs; the two are dealt with separately here.

Section 6. outlines the complete procedure from picture to program, highlighting problems. Supporting information is given in the appendices.

The application programs are written in parallel in Fortran and Cobol. User Control must be written in Fortran.

The sections which follow give examples of an application program and a User Control program using special features. Details which may seem difficult for the user, such as defining add/accumulation fields, defining equivalent fields and combined control, are explained.

A guide to bit/byte manipulation and how to pack and unpack words, and a number of technical details are also given in the following sections. See for ex. 6.7.4.

## 6.2 *CREATING PICTURE FILES*

If several users require access to NSHS software the picture files and application files should be placed on user SYSTEM. The whole procedure is a job for the system supervisor. User SYSTEM should not be used for developing software; the job should be carried out in a public user area and the final tested package copied to user SYSTEM.

A user named NSHS, with 1000 pages, is created for development work.

One source file named PIC-FILE:PIC and one object file named PIC-FILE:OBJ are needed. For the purposes of this illustration they have 10 pages each.

When determining file size it helps to estimate the size of the pictures. Usually the exact layout and final number of pictures will not be known in advance. Picture volume depends mainly on the number of fields and amount of control used and may vary greatly. The source picture takes about 25% more space than the object picture. A rough estimate is 2 blocks of 256 words = 1/2K words for each picture.

The maximum number of pictures on a picture file is 49 and the space required for the source file in this case would be 25 pages.

The commands used to create the picture files mentioned above are:

```
@CREATE-FILE  pic-file:pic,10  CR
```

and

```
@CREATE-FILE  pic-file:obj,10  CR
```

## 6.3 *EXTENDING THE FILE SIZE*

It is no problem to extend the file size if the original estimate was too small. A new file of the desired size is created and the old file copied to it. Two files cannot have the same name; following the renaming procedure as shown below and in Figure 6.1 it is possible to keep the old filename.

If the file is filled whilst a picture is being created, the picture can be saved by the command RESCE-PICTURE.

The commands for extending the picture file are given below along with parameters and comments. Assume that the original source and object files, PIC-FILE:PICT and PICT-FILE:OBJ, were created with 5 pages each and are being extended to 10 pages.

```
SPM*CLOSE-FILES CR

SPM*@CREATE-FILE new-file:pict,10 CR

SPM*@CREATE-FILE new-file:obj,10 CR
SPM*@COPY-FILE new-file:pict,pict-file:pict CR

SPM*@COPY-FILE new-file:obj,pict-file:obj CR

SPM*RESCUE-PICTURE

SPM*DEL-PIC CR

SPM*DUMP-PICT CR

SPM*DESC-PICT CR

or
@DATA-ENTRY-EDITOR CR
@DEL-FILE  pict -fil:pict CR

@DEL-FILE  pict-file:obj CR


@RENAME-FILE  new-file:pict,pict-file:pict CR

@RENAME-FILE  new-file:obj,pict-file:obj CR
```

Picture file exhausted during picture creation. Close the files.
The new files are created with 10 pages each.
Instead of copying the object file, the source file can be compiled using SPM*comp-file in Scr-Def.
Check that the pictures were copied correctly by using the Scr-Def commands, or using Data Entry if available.

The old names are kept for the new files, the old ones must be deleted or renamed. Check the new files first!
The new extended files are now established with the correct names.

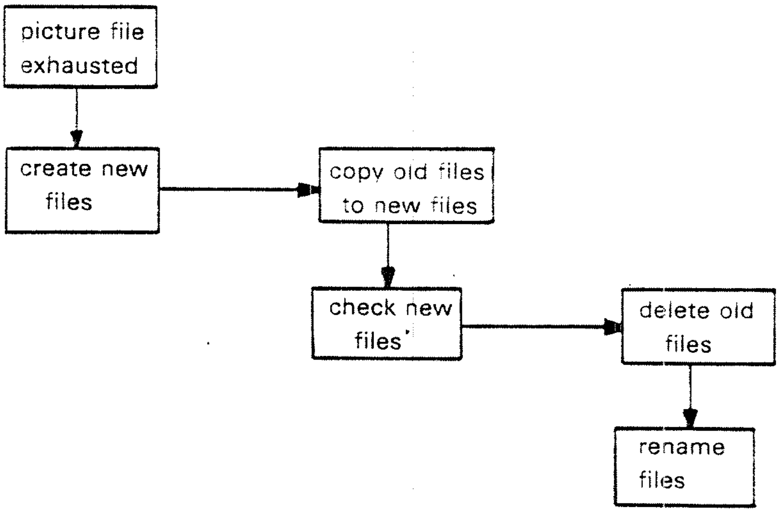


*Figure 6.1: The procedure of extending a picture file.*

## 6.4 *CREATING A PICTURE*

### 6.4.1 *The Relation between Storage Code, Edit Code and Record Format*

When we start creating a picture, we are asked questions about storage and edit code. These codes affect the record format, as explained in the following.

In NSHS there are 5 different storage types (1-5). They are integer, double-integer, byte, Fortran I5 and BCD. We also distinguish between numeric fields and character string fields. Whether or not a field is numeric, also depends on the Edit code. The codes 0-9 and A-J can only be used for numeric fields, while K-S give string fields which may be numeric, alphabetic or alphanumeric.

THE STORAGE CODES INTEGER, DOUBLE INTEGER, FORTRAN I5 AND BCD

These codes can only be used for numeric fields. For these fields, the value displayed on the screen directly reflects the contents of the data element.

STORAGE CODE BYTE (= 3)

The byte fields may be numeric or string type.

*Numeric* byte fields have their value right justified in the data element, and the leading bytes filled with the ASCII value for spaces (40 oct). If the Cobol flag is set (see ITERM(3), section 3.3.3 and 6.7.4.1), the leading 'blanks' will be set to 0 (60 oct).

The Edit code plus the Fill code determine how the field is displayed on the screen, and the 'numeric' how it is stored in the data element. For numeric bytes we distinguish between 'blank' and 'zero' (=0). If the value is zero, the least significant digit (= no 0) is 0. The other bits will be 0 or blanks, depending on the Cobol flag. If the field does not contain any defined value, all the bits are blanks.

*String* byte fields are handled in a different way. The fields that are left justified on the screen, are also left justified in the data element (Edit codes K, L, M, N, O and P), and correspondingly for the right justified fields (Edit codes Q, R and S). Unused positions are filled with the defined 'pad' character which is also inserted in the data element.

See also Appendix A.

## 6.4.2 *Example 1*

### *Creating the picture PER-CX (fig. 1.1)*

The examples have been chosen to demonstrate a reasonable number of typical NSHS features rather than to reflect cases in real life. This example describes the creation of PER-CX shown in Figure 1.1 in the Introduction.

Sections 2.1 and 2.5.2 describe the first part of the procedure, but the commands are repeated here for the sake of completeness. Commands and parameters in Scr-Def must be in CAPITALS.

```
@SCR-DEF   CR
 .
 .
SPM*   CRE-PIC   CR

GIVE SOURCE PICTURE FILE NAME AND TYPE: P-F:P   CR
NOT A PICTURE FILE. IF IT IS TO BE SO TYPE Y: Y   CR

GIVE OBJECT PICTURE FILE NAME AND TYPE: P-F:O   CR
NOT A PICTURE FILE. IF IT IS TO BE SO TYPE Y: Y   CR
```

Answer with abbreviated names for PIC-FILE:PIC and PIC-FILE:OBJ. The message NOT A PICTURE FILE is given the first time the file is used.

If you have forgotten to create the files you must return to SINTRAN and do that. Press the break key. See Section 6.2 Creating Picture Files.

The screen is now cleared; line numbers are displayed along the left border, dots in columns 10, 20 etc. and information appears on the bottom line (24) about number of lines (23) and number of characters per line (79). If the dots are incorrectly distributed you probably gave a wrong terminal number.

To escape from the definition mode and return to Scr-Def press control/F. This means that the picture will not be saved. If you want to save it use control/W and see Section 6.5.

The cursor is placed at the top left of the screen. Press CR to move down one line. Start with underlining leading text ie. PER-CX, by pressing control/U followed by the text; terminate underlining with control/N.

Underlining steals one screen position. Any text below which is not underlined must start in column 2 if it is to line up.

Space forward a few positions and type the text "for person register" in lower case letters.

If you don't succeed in getting lower case letters on the terminal, check if any CAPS switch ot key is set.(On Tandberg VDU 2115 the switch is at the top right of the keyboard, on VDU 2215 the CAPS key is to the left.)
In SINTRAN capital letters can be reset by the command
@TERM-MODE n,,,CR

To proceed, press CR twice and advance one space to start in column 2. Write the text "BDATE:". This is the point where you define a datafield.

Press control/E. The cursor jumps to the bottom line and questions are asked about storage, edit, fill and sign suppress codes, the number of significant characters and control type. See Section 2.2.2 and the following sections.

Avoid typing errors. Take your time and carefully watch the cursor position and question or information displayed. In some cases the cursor jumps between the current field and bottom line.

If you make a mistake whilst editing a field, continue to terminate the field. Then reposition the cursor and redefine the field.

When field definition is terminated, the field is filled with 9 if a numeric field, X if alphanumeric, A if alphabetic and with editing characters according to the code used. The cursor is placed immediately after the field defined and you can proceed to the next leading text or datafield.

After studying section 2.5.4, the questions for defining datafields should cause no problems. Questions connected with field control may be more difficult, especially those concerned with control no. 6, 7 and 10.

Table 6.1 shows the parameters used for creation of PER-CX.

### 6.4.3 *Example 2*

This example describes creation of the picture in Figure 6.2 called INV for invoice. This picture has several datafields on the same line and this line is repeated four times. On the last line TOTAL is added. Lower down the screen there is some information about picture name and filename.

Several pictures may resemble one another or be identical except for defined controls, so it is important to have some identifying text in the picture itself.

The first part of the procedure is the same as Example 1. Continue with control/U, the leading text INVOICE and control/N. It is practical to place field headings after defining the datafields because you can see where to place them, so reserve a few lines for headings and define the first datafield, which is a numeric with 5 digits.

Parameters for this picture are given in table 6.2. Choice of parameters may be governed by practical considerations or sometimes simply by personal preference. In this case the field NO is defined as a single integer. The maximum value must be defined because an integer can hold a maximum of 32766. (This would not be necessary if the field had been defined as a double integer.)

Hyphens (-) are not accepted in alphabetic fields.

When the first line of datafields is defined it is copied to the next one by pressing CR to come up to the beginning of the next line, followed by control/R. The whole line is copied including the parameter definitions. Repeat this process until the line is repeated four times, then define the field for TOTAL separately.

Each record comprises 26 data fields where the SUM fields are no. 5, 10, 15, 20, 25 and the last field, TOTAL, is 26. The accumulation option is used, adding the sum fields together and placing the result in TOTAL. All fields concerned with accumulation must be defined with control = 6. (See Sections 2.5.5.6 to 2.5.5.8.)

Table 6.1  Parameters for picture PER-CX.

Picture name: PER-CX

| Codes | Bdate | Bno | Person name | Person address | Sex | Hourly pay | Dept | Reserve |
|---|---|---|---|---|---|---|---|---|
| storage code | 3 | 3 | 3 | 3 | 3 | 1 | 3 | 3 |
| edit code | C | K | L | M | L | 2 | L | M |
| fill code | 0 | 0 | (0) | (0) | (0) | 0 | (0) | (0) |
| sign suppress code | 1 | - | - | - | - | 1 | - | - |
| no of sign. characters | 6 | 5 | 26 | 30 | 1 | 5 | 6 | 12 |
| control type | 8 | 0 | 9 | 0 | 10 | 10 | 2 | 1 |

Parameters for the control types are given below:

| text on the screen | parameters | comments |
|---|---|---|
| | BDATE | datafield |
| CONTROL TYPE | 8 | system control, see 2.5.5 |
| SYSTEM CONTROL NUMBER (1-255) | 5 | this corresponds to day, month, year; see appendix D |
| DATE CHECK NUMBER (1-6) | 5 | date must be before today; if a more specific check is required, User Control must be used in addition |
| | SEX | datafield |
| CONTROL TYPE | 10 | controls 2 and 9 are used here |
| NO OF COMBINED CONTROLS (from 2-40) | 2 | |
| CONTROL TYPE | 2 | legal values to be defined |
| NO OF LEGAL VALUES | 2 | two values are legal |
| FIRST VALUE | M | cursor jumps to the datafield to receive the value; M for male is given |
| TYPE ANY CHARACTER THEN GIVE NEXT LEGAL VALUE | 'Sp'<br>F | M disappears and the field receives the new value F for female |
| .OR./.AND.NEXT CONTROL (1 OR 0) | 0 | both controls, .AND. |
| CONTROL TYPE | 9 | User Control |
| USER CONTROL NUMBER (1-155) | 3 | the other two fields in the picture, person name and hourly pay, were given 1 and 2 |

| | | |
|---|---|---|
| datafield | Hourly Pay | |
| CONTROL TYPE | 10 | controls 4 and 9 will be used |
| NO OF COMBINED CONTROLS (FROM 2 TO 40) | 2 | |
| CONTROL TYPE | 4 | legal range |
| LOWEST LEGAL VALUE | 10.00 | the cursor is back in the field to receive a value; 10 CR is acceptable |
| TYPE ANY CHARACTER THEN | 'sp' | the maximum value; the cursor is positioned after the field. The procedure now is as described for SEX, except User control no = 2. |
| GIVE HIGHEST LEGAL VALUE | 250.00 | |

| | | |
|---|---|---|
| datafield | DEPT | |
| CONTROL TYPE | 2 | the procedure is like that for SEX |
| NO OF LEGAL VALUES | 5 | |
| FIRST VALUE | ACC | abbreviation for accounting |
| TYPE ANY CHARACTER THEN | 'sp' | |
| GIVE NEXT LEGAL VALUE | ENG | engineering |
| . | 'sp' | |
| . | FIN | financial |
| . | 'sp' | |
| . | STO | store |
| . | 'sp' | |
| . | TRA | transport |

| | | |
|---|---|---|
| datafield | RESERVE | |
| CONTROL TYPE | 1 | default value inserted if no other value is given |
| DEFAULT VALUE | *** | the field is defined as alphanumeric |

Picture design is now completed. The picture must be written to the picture file to be stored. This is explained in Section 6.5.

To redefine a datafield or change a leading text, move the cursor to the correct place on the screen using the arrow keys. Take care with CR: everything on a line behind the cursor is deleted when CR is pressed.

Table 6.2 Parameters for picture INV

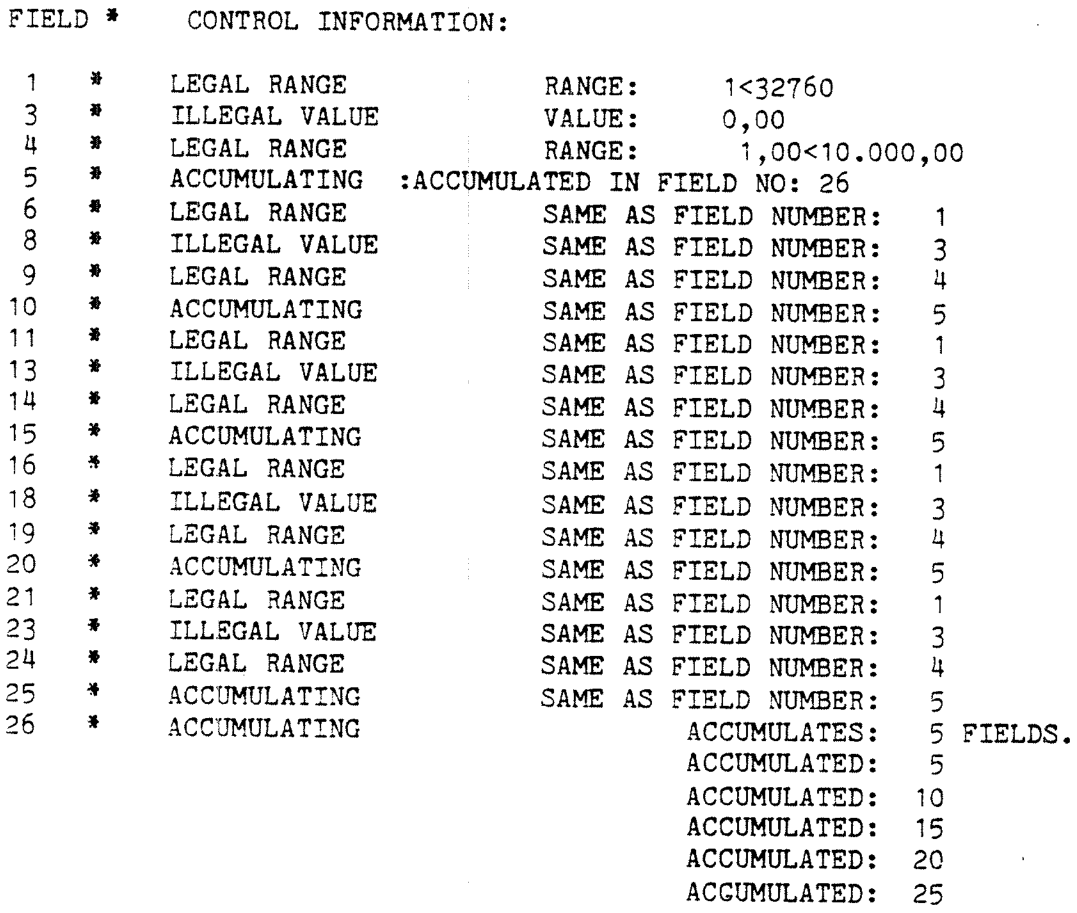
Picture name: INV

| Codes | No | Text | Amount | Un-pr | Sum | Total |
|---|---|---|---|---|---|---|
| store code | 1 | 3 | 2 | 2 | 2 | 2 |
| edit code | A | M | 2 | 2 | 2 | 2 |
| fill code | 0 | - | 0 | 0 | 0 | 0 |
| sign suppress | 1 | - | 0 | 1 | 0 | 0 |
| sig. characters | 5 | 13 | 6 | 7 | 8 | 9 |
| control type | 3 | 0 | 4 | 4 | 6 | 6 |
| no of illegal values | | | 1 | | | |
| illegal value | | | 0 | | | |
| lowest legal value | 1 | | | 1.00 | | |
| highest legal value | 32760 | | | 10000.00 | | |
| accumulated field no or 0 if none | | | | | 26 | 0 |
| no of fields to be accumulated or 0 if none | | | | | 0 | 5 |
| accumulated field number | | | | | | 5<br>10<br>15<br>20<br>25 |

Parameters requested under the different control numbers are also given here. How the picture is saved (stored) in the picture file is explained in the next section.

```
 INVOICE

  NO      TEXT          AMOUNT    UN.PR          SUM             TOTAL

 99999 XXXXXXXXXXXXX 9.999,99- 99.999,99   999.999,99-
 99999 XXXXXXXXXXXXX 9.999,99- 99.999,99   999.999,99-
 99999 XXXXXXXXXXXXX 9.999,99- 99.999,99   999.999,99-
 99999 XXXXXXXXXXXXX 9.999,99- 99.999,99   999.999,99-
 99999 XXXXXXXXXXXXX 9.999,99- 99.999,99   999.999,99-   9.999.999,99-


*** picture file: PIC-FILE   picture name: INV ***
    no user control


PICTURE HAS 23 LINES, 79 CHARACTERS PER LINE, AND DISPLAY MODE 1

FIELD*LINE*COL.* ST.CO*FILL*     EDIT CODE     *SIGN*NUMB*CONTROL

  1   * 6  *  2 * INT   *    * DIGIT   - R.ADJUST*     *  5 *    4
  2   * 6  *  8 * BYTE  *    * STRING  - ALPNUM-L*     * 13 *    0
  3   * 6  * 22 * D.INT*     * DEC.PNT-,2        * S   *  6 *    3
  4   * 6  * 32 * D.INT*     * DEC.PNT-,2        *     *  7 *    4
  5   * 6  * 44 * D.INT*     * DEC.PNT-,2        * S   *  8 *    6
  6   * 7  *  2 * INT   *    * DIGIT   - R.ADJUST*     *  5 *    4
  7   * 7  *  8 * BYTE  *    * STRING  - ALPNUM-L*     * 13 *    0
  8   * 7  * 22 * D.INT*     * DEC.PNT-,2        * S   *  6 *    3
  9   * 7  * 32 * D.INT*     * DEC.PNT-,2        *     *  7 *    4
 10   * 7  * 44 * D.INT*     * DEC.PNT-,2        * S   *  8 *    6
 11   * 8  *  2 * INT   *    * DIGIT   - R.ADJUST*     *  5 *    4
 12   * 8  *  8 * BYTE  *    * STRING  - ALPNUM-L*     * 13 *    0
 13   * 8  * 22 * D.INT*     * DEC.PNT-,2        * S   *  6 *    3
 14   * 8  * 32 * D.INT*     * DEC.PNT-,2        *     *  7 *    4
 15   * 8  * 44 * D.INT*     * DEC.PNT-,2        * S   *  8 *    6
 16   * 9  *  2 * INT   *    * DIGIT   - R.ADJUST*     *  5 *    4
 17   * 9  *  8 * BYTE  *    * STRING  - ALPNUM-L*     * 13 *    0
 18   * 9  * 22 * D.INT*     * DEC.PNT-,2        * S   *  6 *    3
 19   * 9  * 32 * D.INT*     * DEC.PNT-,2        *     *  7 *    4
 20   * 9  * 44 * D.INT*     * DEC.PNT-,2        * S   *  8 *    6
 21   *10  *  2 * INT   *    * DIGIT   - R.ADJUST*     *  5 *    4
 22   *10  *  8 * BYTE  *    * STRING  - ALPNUM-L*     * 13 *    0
 23   *10  * 22 * D.INT*     * DEC.PNT-,2        * S   *  6 *    3
 24   *10  * 32 * D.INT*     * DEC.PNT-,2        *     *  7 *    4
 25   *10  * 44 * D.INT*     * DEC.PNT-,2        * S   *  8 *    6
 26   *10  * 58 * D.INT*     * DEC.PNT-,2        * S   *  9 *    6
```

*Figure 6—2a: Picture description for INV.*



```
FIELD *    CONTROL INFORMATION:

  1   *   LEGAL RANGE                RANGE:      1<32760
  3   *   ILLEGAL VALUE              VALUE:      0,00
  4   *   LEGAL RANGE                RANGE:       1,00<10.000,00
  5   *   ACCUMULATING  :ACCUMULATED IN FIELD NO: 26
  6   *   LEGAL RANGE                SAME AS FIELD NUMBER:   1
  8   *   ILLEGAL VALUE              SAME AS FIELD NUMBER:   3
  9   *   LEGAL RANGE                SAME AS FIELD NUMBER:   4
 10   *   ACCUMULATING               SAME AS FIELD NUMBER:   5
 11   *   LEGAL RANGE                SAME AS FIELD NUMBER:   1
 13   *   ILLEGAL VALUE              SAME AS FIELD NUMBER:   3
 14   *   LEGAL RANGE                SAME AS FIELD NUMBER:   4
 15   *   ACCUMULATING               SAME AS FIELD NUMBER:   5
 16   *   LEGAL RANGE                SAME AS FIELD NUMBER:   1
 18   *   ILLEGAL VALUE              SAME AS FIELD NUMBER:   3
 19   *   LEGAL RANGE                SAME AS FIELD NUMBER:   4
 20   *   ACCUMULATING               SAME AS FIELD NUMBER:   5
 21   *   LEGAL RANGE                SAME AS FIELD NUMBER:   1
 23   *   ILLEGAL VALUE              SAME AS FIELD NUMBER:   3
 24   *   LEGAL RANGE                SAME AS FIELD NUMBER:   4
 25   *   ACCUMULATING               SAME AS FIELD NUMBER:   5
 26   *   ACCUMULATING                        ACCUMULATES:   5 FIELDS.
                                              ACCUMULATED:   5
                                              ACCUMULATED:  10
                                              ACCUMULATED:  15
                                              ACCUMULATED:  20
                                              ACCUMULATED:  25
```

*Figure 6—2b: Picture description for INV (cont.).*

## 6.5 *TERMINATING A PICTURE*

If you are satisfied with the picture and want to store it on the picture file, in this case PIC-FILE, press control/W.

You are then asked the following questions (the operator's input is in italion)

```
PICTURE NAME: PER -CX  cr
SOURCE PICTURE PER-CX STORED
SOURCE PICTURE LENGTH 425 WORDS
LIST FILE:  cr
OBJECT PICTURE PER-CX STORED
OBJECT PICTURE LENGTH 330 WORDS
```

This is the normal output when the compilation is successful. If the compiler detects an error, the message OBJECT PICTURE....NOT STORED appears on the screen. In this case LIST FILE should be L-P (line-printer) or the standard name of some device giving a hard copy for the error report.

If you do not want to store the picture, press control/F to return to Scr-Def.

## 6.6 *MODIFYING A PICTURE*

It may be necessary to change layout following some change to the source document, or to change leading text or redefine a field in the picture. Any change affecting leading text or record layout changes has no further consequences. Changes to datafields affect the corresponding record files and application programs using that picture may have to be revised. The NSHS supervisor must have full control of pictures and application programs. Modifications should be carried out before being implemented on user SYSTEM.

Figure 6.3 shows the procedure for picture modification. Enter Scr-Def and give the command

```
SPM* MODIFY-PICTURE
```

Questions are asked about source and object picture file, including type. Abbreviations can be used.

You are then asked about picture name which cannot be abbreviated; the maximum number of characters for the name is 8.

If an error is detected, a message appears on the screen. The error may result from a typing mistake or reveal something more serious. Eg.:

File not continuous
A continuous file is required for picture files. This error occurs when files are copied from diskette to disk without creating the receiving file in advance. See Sections 1.2 and 6.2.

No such filename
Probably due to misspelling. Check the filename by @LIST-FILE.

No source picture called....
Probably due to misspelling. Remember picture names must not be abbreviated. This error may occur when no datafields are defined in the picture. Use SPM* DEL-PIC to list existing pictures on the file.

No object picture called....
An error was detected during compilation and the picture was never stored in object form; or the picture has been deleted from the object file. Use SPM* DEL-PIC to check.

When the file and picture names are accepted, the picture is displayed with the cursor at the top left of the screen. Use the arrow keys to move the cursor and take care with CR because everything behind the cursor will be erased.

When modifying a picture, the same editing functions are used as when creating a picture. When modification is complete, press control/W. You are asked "Is destination to be different from source?" (Y for Yes). If the modified picture is to keep the same name, answer N and the rest of the procedure takes place automatically. Press CR for LIST-FILE to get a report on the screen; normally this is only a few words. If the picture has many fields or there are several errors, repeat the sequence and give L-P or another device to give a hard copy to study.

It is often convenient to create a new picture (with a new name) by modifying a similar old picture. Eg. PER-C and PER-CX were created using PER and redefining the fields which differ by User Control.

An element of security is built into the modification procedure. Eg. if a new picture is desired, it is not accepted if the name already exists. Similarly if you intend to use an old picture, an error message is given if that picture is not found.

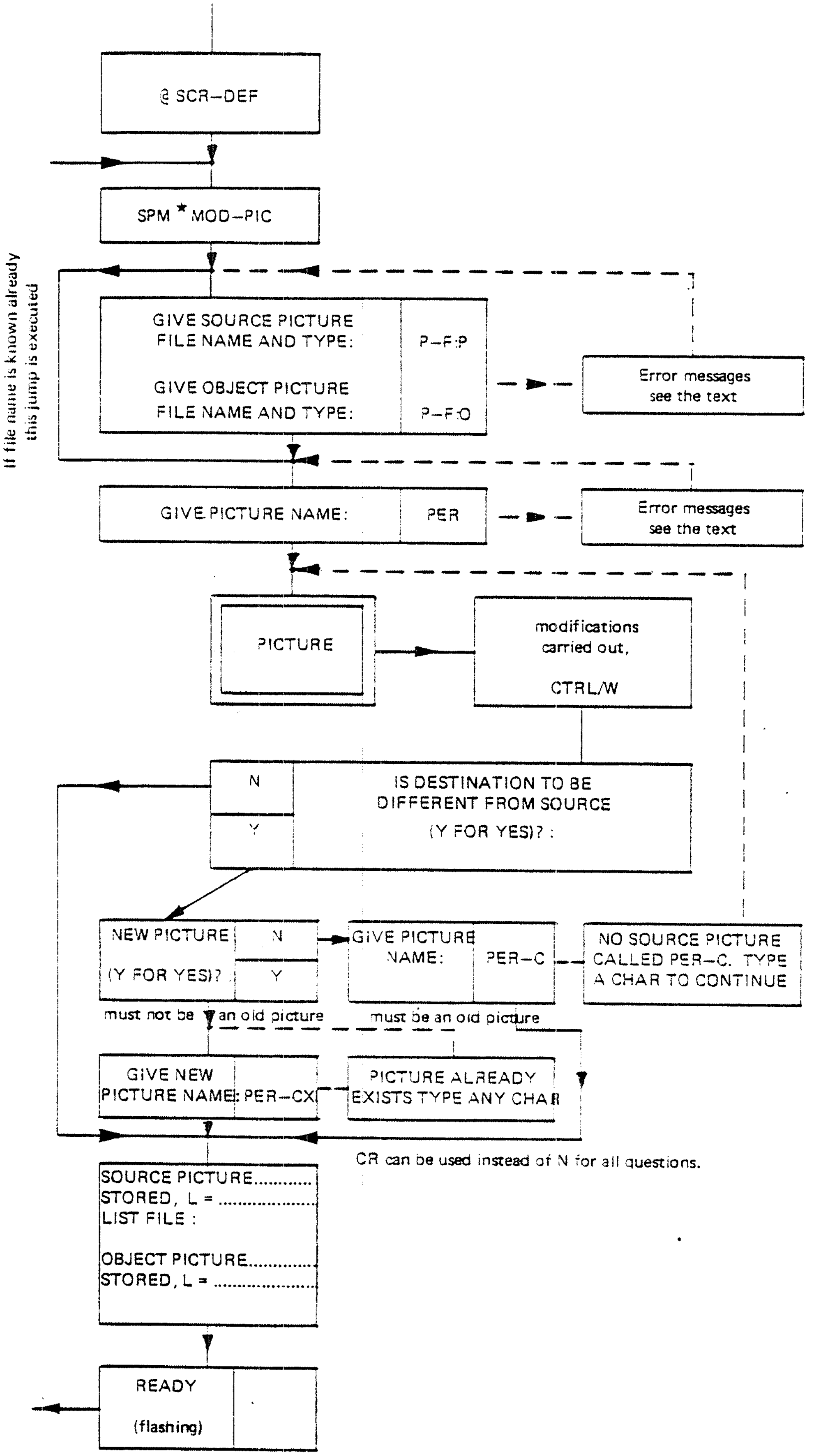

@ SCR–DEF
SPM * MOD–PIC
If file name is known already this jump is executed
GIVE SOURCE PICTURE FILE NAME AND TYPE:
P–F:P
GIVE OBJECT PICTURE FILE NAME AND TYPE:
P–F:O
Error messages see the text
GIVE PICTURE NAME:
PER
Error messages see the text
PICTURE
modifications carried out,
CTRL/W
N
Y
IS DESTINATION TO BE DIFFERENT FROM SOURCE (Y FOR YES)? :
NEW PICTURE (Y FOR YES)?
N
Y
GIVE PICTURE NAME:
PER–C
NO SOURCE PICTURE CALLED PER–C. TYPE A CHAR TO CONTINUE
must not be an old picture
must be an old picture
GIVE NEW PICTURE NAME:
PER–CX
PICTURE ALREADY EXISTS TYPE ANY CHAR
CR can be used instead of N for all questions.
SOURCE PICTURE............ STORED, L = ...................
LIST FILE :
OBJECT PICTURE............ STORED, L = ...................
READY (flashing)

*Figure 6—3: Flow Diagram for the Command SPM* MOD-PIC*

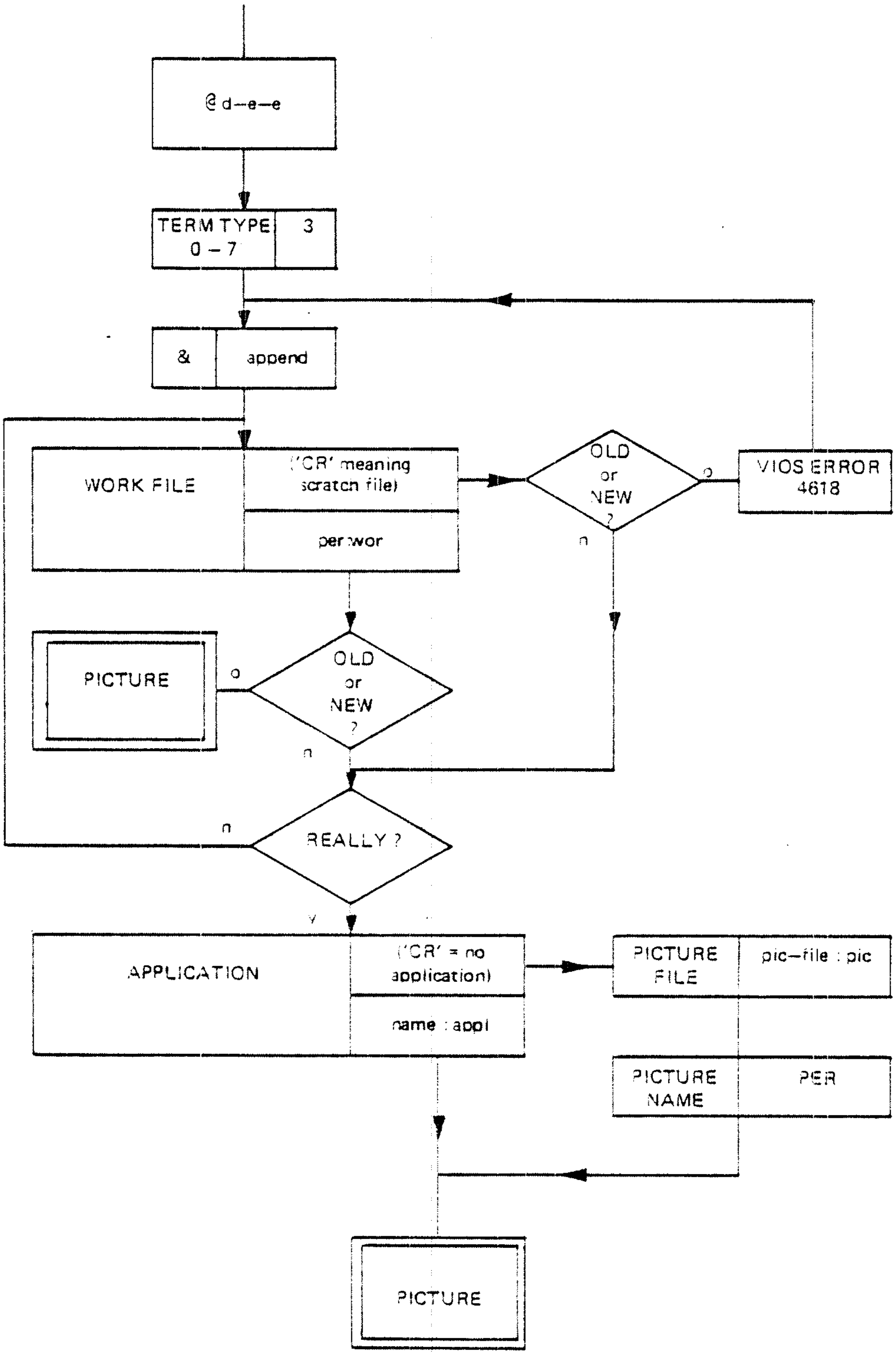


*Figure 6—4: Calling a picture in Data Entry*

## 6.7 *APPLICATION PROGRAMMING*

### 6.7.1. *Introduction*

Application programs are generally used to handle data entered by and displayed on terminals (VDUs).

The basic element is the *picture* which must exist on a picture file. The creation of pictures is described in sections 2.5 and 6.4.

Application programs using NSHS contain several subroutine calls with parameters. The programming language may be Fortran or Cobol or any language that can call a Fortran subroutine. These subroutines are found in the NSHS library.

In the heading of the program it is also necessary to define Common areas and data arrays containing special data values.

In Cobol programs this is accomplished by socalled Data blocks, which are loaded together with the main program. This is explained under the Cobol program examples.

In ND's Cobol, octal numbers are not accepted, and must be converted and represented on decimal form.

## 6.7.2. *ITERM IPRIV and IPUBL*

As mentioned, special data areas must be defined in the program heading. For convenience, following 'standard' names are used.

The common lay-out is:

```
COMMON/PRIVATE/ITERM(128),IPRIV(1920)
COMMON/PUBLIC/IPUBL(6)
.
.
.
DATA  ITERM/1,5,7B,3,20000B,0,0,0,0,1920,118*0/
DATA  IPRIV/0,1920,8,0,0,1915*0/
```

The figure in the ( ) is the number of words allocated for the arrays in this example. For the sake of space economy, the sum should be an integer number of Sintran pages. Here 128+1920=2048 words equal 2 pages.

If only Privat pictures are used, the min value of IPUBL should be 6 (as used here). A rough estimate for IPRIV = sum of object pictures used + 100. This number + 128 for ITERM is rounded up to make an integer number of Sintran pages, as mentioned above.
If only Public pictures are used, the min value of IPRIV should be 30. The same rule applies to IPUBL as used for IPRIV above.

The data values for ITERM and IPRIV above are determined partly by system and external devices, partly by the programmer's preference.

The data fields are described in section 3.3—3.4 but are repeated here for convenience:

DATA ITERM

ITERM ( 1) Logical device no.

.. ( 2) Terminal type./Picture Displacement

.. ( 3) Cobol Flag/Defined escape character.

.. ( 4) Read strategy.

.. ( 5) Pad char/Program mode.

.. ( 6) Cursor position.

.. ( 7) Error code.

.. ( 8) Break/Echo strategy.

.. ( 9) Status/Option word.

.. (10) Length of IPRIV.

DATA IPRIV

IPRIV (1) Fixed value = 0.

.. (2) Length of IPRIV (= ITERM(10)).

.. (3) Max number of picture descriptions (max = 8).

.. (4) Picture number actually in use. Initially = 0.

.. (5) Reserved. Initiation value = 0.

The rest of the arrays not holding any initial value is used as buffer area by NSHS, and must be set to 0. Numbers followed by 'B' are interpreted as octal numbers. A few of the data are described more in detail in section 6.4.

## 6.7.3. *Basic Calls*

All application programs using NSHS must start with the important calls GTPIC (Get picture) and GTFDN (Get field numbers). The first call introduces the names of the pictures to be used and is usually carried out only once during program execution.

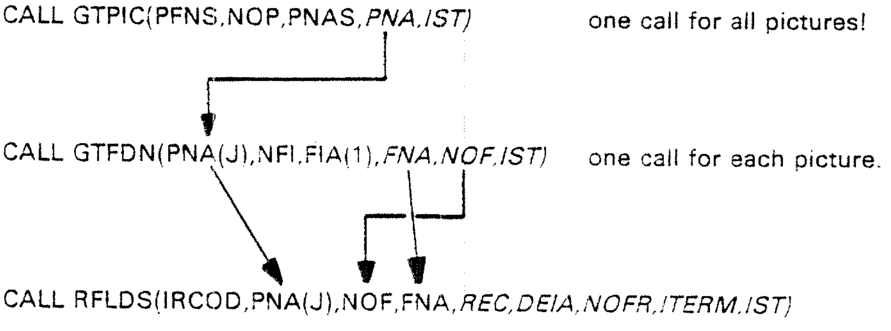
The second call transforms information about the fields in the picture to an accessible form and is repeated for each picture to be used. See fig. 6-12.

The 'heart' of NSHS is the call RFLDS (Read fields) which is used for entering new data, verifying old data, checking old records on data files etc. Apart from these basic calls are several calls for different purposes.

See the list of available calls, section 3.6.2.

The calls comprise a list of parameters. The first ones are input parameters, the others (in italian typing) receive the result of the call. An important point here is that the programmer does not have to give values to all input parameters. The NSHS system itself will pick the necessary information, if available, and set values for the input parameters.

In the following, the calls for GTPIC, GTFDN and RFLDS with parameters are shown. The transport of information from one call to another is indicated by arrows. The abbreviated Fortran 'standard' names have been used. The full names of the data fields are found in the table section 3.6.3.15.



CALL GTPIC(PFNS,NOP,PNAS,*PNA,IST)* one call for all pictures!

CALL GTFDN(PNA(J),NFI,FIA(1),*FNA,NOF,IST)* one call for each picture.

CALL RFLDS(IRCOD,PNA(J),NOF,FNA,*REC,DEIA,NOFR,ITERM,IST)*

For example, the picture-number-array PNA which is a receiving field in the first call, gives input to the next call. The value of J is set in the program, and often IPN is used instead: IPN=PNA(J).

When an application program is planned, it is wise to make a flow- chart for the program steps and an extract of the calls as demonstrated for program YSFPER section 6.7.5.2 and fig.6-13.

It is important to notice here that output parameters from one call is used as default input parameters to the next one. But at the same time the programmer has full control and may change the parameter values if required. Thus the number of fields NOF, which is initally 0, is obtained from the call GTFDN counting the number of fields in the picture. This number is later taken as input to the call RFLDS, if something else is not required. See the calls in the text above.

## 6.7.4. *Bit and Byte Manipulation*

One computer word consists of 2 bytes a 8 bits. Each bit can hold the value 0 or 1. In some cases the value of a word is built up by independant byte values. In Fortran it is also possible to work with commands on bits and bytes, and this technique is frequently applied in the User control programs. In the following both cases are demonstrated.

### 6.7.4.1. Cobol Flag/Escape Character = ITERM(3)

See section 3.3.3. In this case the left byte is used for the Cobol flag, the right for the Escape character value.

Cobol flag: By setting bit no 15 to 1, the Cobol flag is set, and the processing will follow the special rules for Cobol programs.

Escape char: The bits no 0, 1 and 2 of the right byte are used for the Escape character. Any character the keyboard can be used, but Control/G is frequently used because it is standard in the Data Entry system. The control key means subtracting 100B from the ASCII key value. (Octal numbers)

Example 1.
fig.6-5

Fortran program: bit no 15 = 0.
Escape char: Control/G

G = 107B (ASCII octal value)
Control = -100B

Contr/G = 7B = 111 binary

The binary values are shown in fig. 6—5.
The value of the word ITERM(3) = 7B.

Example 2.
fig.6-6

Cobol program: bit no 15 = 1.
Escape char: Control/@

@ = 100B (ASCII octal value)
Control = -100B

Contr/@ = 000B

The binary values are shown in fig. 6—6.
The value of ITERM(3) = 100000B.

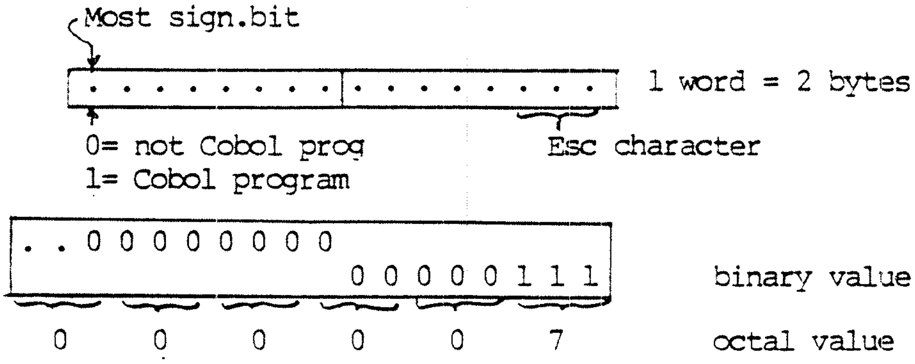


*Figure 6—5: Example 1 for ITERM(3)*

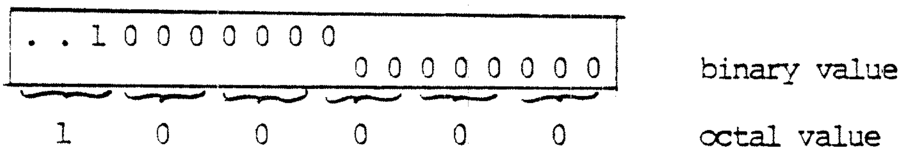


*Figure 6—6: Example 2 for ITERM(3)*

### 6.7.4.2. Pad Character/Program Mode = ITERM(5)

See section 3.3.5. The left byte is used for Pad character, the right byte for Program mode.

| | |
|---|---|
| Pad character: | Any character will do. In most cases 'SP' or 0 is used. |
| Program mode: | If Private picture: Progr mode = 0.<br>If Public picture : Progr mode = 1. |

Example 3.
fig.6-7

Pad character: 'SP' = 40B (ASCII octal value)
= 100 000 binary

Program mode: Private picture
Bit no 0 = 0.

The binary representation is shown in fig. 6—7.
The value of ITERM(5) = 20000B.

Example 4.
fig.6-8

Pad character: 0 = 60B (ASCII octal value)
= 110 000 binary

Program mode: Public picture
Bit no 0 = 1.

The binary representation is shown in fig 6-8.
The value of ITERM(5) = 30001B.

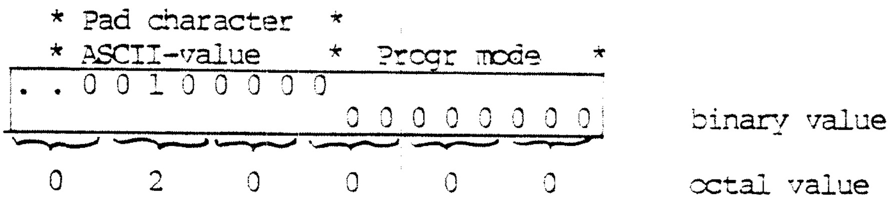


*Figure 6—7: Example 3 for ITERM(5)*

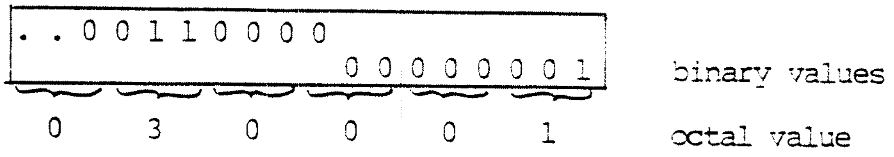


*Figure 6—8: Example 4 for ITERM(5)*

### 6.7.4.3. Extracting Bytes from Words

Words are built up by bits where each bit has a special meaning. It is also sometimes necessary to 'explode' a word to find the value of a bit or a byte.
The values wanted can easily be found by using the operator .AND. in a Fortran statement together with a number N consisting of 0 in the vaste positions, 1 in the positions to be extracted.

*Example 5:* In NSHS the position of the cursor on the screen is found in the word ITERM(6). The *row* no is put in the left byte, the *column* no in the right byte. For a special position of the cursor we want to extract the row and column.

```
ITERM(6) = ISHFT(23,8) + 13 = 13415B where 23 (27B) is the row no,
                                     13 (15B) the column number.
```

ISHFT (N,8) means that the number N is moved 8 pos. to the left; thus being placed in the left byte of the word.

By setting the bits of the left byte to 0, and the bits of the right to 1, which means the value 377B, and using this with .AND., the column no is extracted. See fig. 6-9.The complete statements are:

```
IP  = ITERM(6)
ICOL= IP.AND.377B
IROW= ISHFT(IP,-8)
```

```
. . 0 0 0 1 0 1 1 1                       27B
                    0 0 0 0 1 1 0 1       15B

. . 0 0 0 0 0 0 0 0                       0
                    1 1 1 1 1 1 1 1       377B

. . 0 0 0 0 0 0 0 0
                    0 0 0 0 1 1 0 1       15B
```

*Figure 6—9,10: Extracting row and column number from curser position 13415B.*

The technique of handling bits and bytes in programs is frequently used in writing the User control programs. For further information see the Fortran manuals.

## 6.7.5. *Program Examples*

In the following, examples are given on simple background programs using PRIVAT pictures. This will do in most cases. RT programs, and programs using PUBLIC pictures will not be considered here. The programs are made as simple as possible to show the NSHS features and are given both in Fortran and Cobol.

Some information is given throughout the programs by comment lines. Additional information is given in the text.

All programs have been tested running on the computer ND-100.

### 6.7.5.1 Data Discipline in Using NSHS

When NSHS is used in programming, perhaps in connection with data bases, a lot of data elements (like files,programs and pictures) are involved. To avoid confusion, it is then necessary to practice a good "data discipline".

Because the picture is the basic element in NSHS, it is wise to refer all names of files and programs to the picture name. The picture name should in turn as good as possible reflect the function of the picture.

It is also recommended to use the standard names for functions and parameters. See section 3.6.3.15. It will then be much more easy to maintain programs, avoid errors, and for ND to give assistance if problems occur.

For example, if the picture PER (for PERson register) is used, the program is called SFPER:SYMB where SF here means Screen-Fortran. If no ambiguity is possible, the prefix letters are skipped. If the picture PER is used for several programs, they must be distinguished by an additional letter. L for loading, R for reading etc.

The output file, the person register file, should simply be called PER:DATA, independant of program language or purpose of processing. See the example in fig. 6-11.

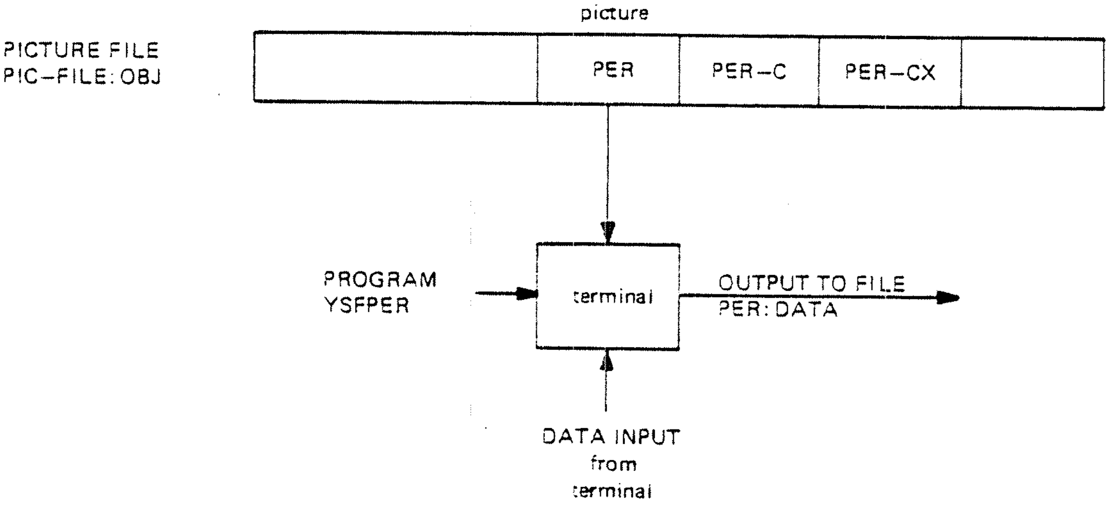


*Figure 6—11: Interaction between inout, picture, program and output*

## 6.7.5.2 Program Example 1: YSFPER/ (F)

This is a Fortran program for entering person data to the file PER:DATA by means of the picture PER residing on the picture file PIC-FILE:OBJ.

There are altogether three pictures named 'PER' on the picture file, PER, PER-C, PER-CX. These pictures are identical except for the *User control.* Picture PER has no user control, while PER-C and PER-CX have control on different combinations of fields. The standard controls defined for each field under NSHS are identical for the three pictures.

The *picture description* for PER is shown in table 6—3.

The description consists of three parts: First the picture as it appears on the screen, then information about the fields in sequential order where the control numbers are found far to the right. From this it can be seen that no User control has been defined (=9), but the field BDATE has got a system check (=8). Finally some more information is given as to the field control. For example, the field SEX has two legal values: M (male) or F (female). See section 2.5.4 to check the information in the picture description.

It is evident that all available *system controls* should be used before any *user control* is applied.

The data are placed on the file PER:DATA. It may be created in advance, or during runtime by the Opening procedure in the program.

## DESCRIPTION OF THE PROGRAM

The principle steps of the program are shown in fig.6—13. The terminal type is set in the program equal to 3 corresponding to Tandberg terminals. It might have been given interactively in the same way as for the PICTURE FILE NAME and TYPE and the PICTURE NAME STRING as shown in the source program below. The numbers in the flow-chart correspond to the labels in the source program. The program is like any other FORTRAN program except for the calls to NSHS, referring to the screen-handling library,called SCREEN-LIB.

The simple task of this program is to receive data entered to the terminal, transmit them to a disk file in the form of records, and write selected data from the record to the line-printer. The disc- file is opened with the WA-option (write-append); the new records will then be added to those already on the file.

```
-----------------------------------------------------------------
PER                              for person register

 BDATE :  99.99.99
 BNO   :  99999

 PERSON NAME:  AAAAAAAAAAAAAAAAAAAAAAAAAA

 PERSON ADDR:  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

 SEX        :  A
 HOURS' PAY:  999,99
 DEPARTMENT:  AAAAAA

 RESERVED:  XXXXXXXXXXXX


 *** Picture file PIC-FILE ***
 *** No user control       ***

PICTURE HAS 23 LINES, 79 CHARACTERS PER LINE, AND DISPLAY MODE 1

 FIELD * LINE * COL. * ST.CO * FILL *       EDIT CODE        * SIGN * NUMB * CONTR

   1   *   4  *  11  *  BYTE *      * DIGIT  - CLOCK        *      *   6  *   8
   2   *   5  *  11  *  BYTE *      * STRING - NUM.-LEFT    *      *   5  *   0
   3   *   7  *  16  *  BYTE *      * STRING - ALPHA.-L     *      *  26  *   0
   4   *   9  *  16  *  BYTE *      * STRING - ALPNUM-L     *      *  30  *   0
   5   *  11  *  15  *  BYTE *      * STRING - ALPHA.-L     *      *   1  *   2
   6   *  12  *  15  *  INT  *      * DEC.PNT-,2            *      *   5  *   4
   7   *  13  *  15  *  BYTE *      * STRING - ALPHA.-L     *      *   6  *   2
   8   *  15  *  13  *  BYTE *      * STRING - ALPNUM-L     *      *  12  *   1

 FIELD *    CONTROL INFORMATION:

  1   *    SYST. DEFINED          DATE(D,M,Y)    LT
  5   *    LEGAL VALUE            VALUE:
                                  M
                                  F
  6   *    LEGAL RANGE            RANGE:  10,00<250,00
  7   *    LEGAL VALUE            VALUE:
                                  ACC
                                  ENG
                                  FIN
                                  STO
                                  TRA
  8   *    DEFAULT                VALUE:  ***
```

*Table 6-3: Picture description for PER*

The five first NSHS calls are almost standard in all NSHS programs. Here all data tables necessary to handle data on the screen are set up. If several pictures are used from a program, all the picture names should be given continously in the PICTURE NAME STRING, where each name should occupy 8 positions.

For example, if the names PER, ORDER, INVOICE are used, the picture name string should be:

PER.....ORDER...INVOICE.

where each dot here represents one space. In the picture number array, PNA(1) will be PER, PNA(2) is ORDER etc.

Usually it is possible to pass a field on the screen without giving any data, i.e just giving 'CR'. The data field will then be a blank or 0 depending on the data type. However, a 'must' directiv may be given to prevent skippingof data. This is accomplished by means of the call ZMUST. Here 'must' is described for the name fields, no 3 and 4. It is of course not necessary to prescribe 'must' on fields where only certain legal values are accepted. For ex. the field SEX will only accept either 'M' or 'F'.

'Clear fields' is used to clear the fields on the screen for old input, which is the normal procedure. Of course the *leading text* is not affected.

Data entered to the terminal will be placed in the record buffer, which is the REC parameter in the calls. And the data in REC will remain unchanged as long as they are not cleared or overwritten by new data. This also means that if fields are passed without giving any input, the old data remain intact, and can be used together with the new input for the new record.

This feature can be utilised when the same data appear in the same field in several consecutive records, as demonstrated in program ZSFPER.

Usually however, old values are cleared.

We then have the call RREAD (corresponding to ZREAD). See section 3.6.4.12. It is used because of the ZMUST call. See also fig 6-12. Each field in the picture is identified by a 16 bits word, where the bits 13, 14 and 15 are reserved for 'MUST', 'LOCK' and 'READ'. In this program the fields 3 and 4 are set 'musted'. Let us see how this works:

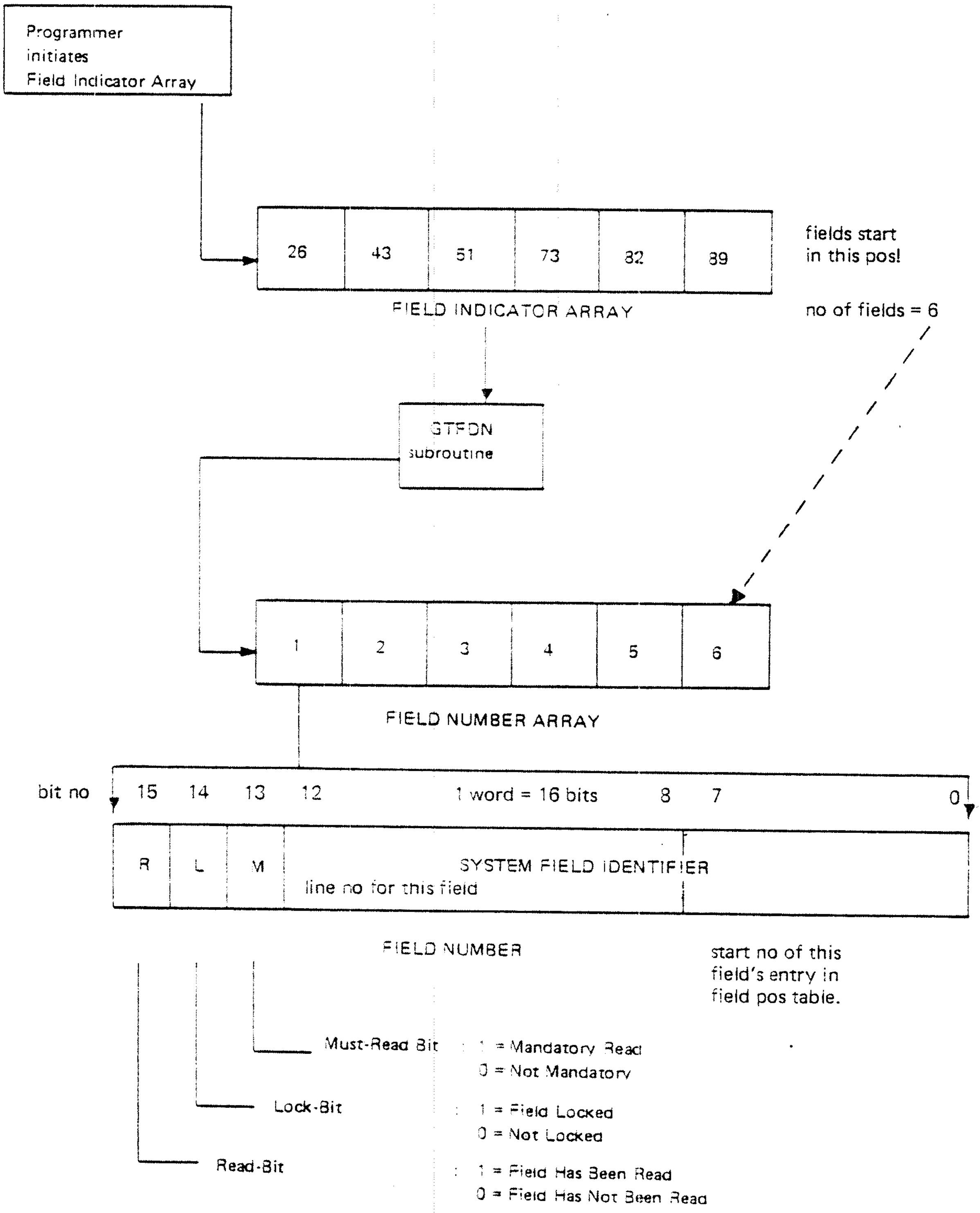


*Figure 6—12: GTFDN subroutine transforms absolute screen positions to field numbers.*

When we enter a value to a field, the ***Read-bit is set = 1.*** When a field is passed without giving any value, the Read-bit is set to 0. At the beginning, all bits are 0. Because of ZMUST on fields 3 and 4, the Read-bits are checked for these fields, and it will not be possible to proceed without giving values to the fields.

However, next time (next record) the Read-bits are set, and it is now possible to skip fields 3 and 4 in spite of the 'must' requirement. The read-bit is reset , and for the next record we have to enter data again. The procedure is repeated, meaning that it is possible to skip the fields 3 and 4 in every second record! We therefore have to remove the Readbits pertaining to the previous record, and this is for convenience done here for all the fields, not only for fields 3 and 4.

The Read-fields call is the 'heart' of the NSHS system. When new data are *read* into the system, the parameter IRCODE=0. The Read- fields call may however be used for several other purposes, and the Reader is invited to exercise the other 4 features described in section 3.6.4.3.

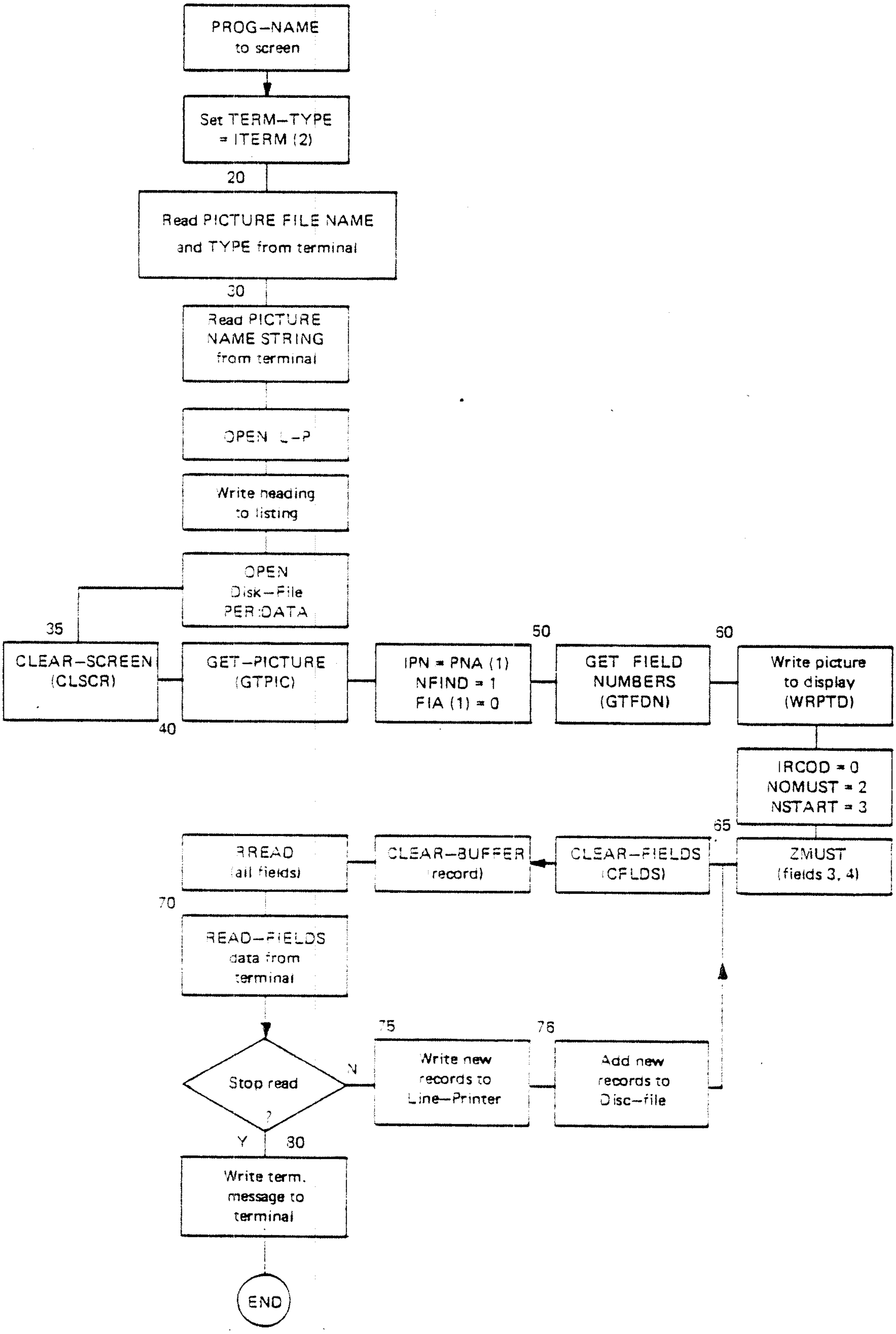

PROG–NAME to screen
Set TERM–TYPE = ITERM (2)
20
Read PICTURE FILE NAME and TYPE from terminal
30
Read PICTURE NAME STRING from terminal
OPEN L–P
Write heading to listing
OPEN Disk–File PER:DATA
35
CLEAR–SCREEN (CLSCR)
40
GET–PICTURE (GTPIC)
IPN = PNA (1) NFIND = 1 FIA (1) = 0
50
GET FIELD NUMBERS (GTFDN)
60
Write picture to display (WRPTD)
IRCOD = 0 NOMUST = 2 NSTART = 3
65
ZMUST (fields 3, 4)
CLEAR–FIELDS (CFLDS)
CLEAR–BUFFER record)
BREAD (all fields)
70
READ–FIELDS data from terminal
Stop read ?
N
Y
75
Write new records to Line–Printer
76
Add new records to Disc–file
80
Write term. message to terminal
END

*Figure 6—13: Flow-chart for program YSFPER*

The registration of data can be terminated by giving the terminating function defined in ITERM(3) from the key-board. Frequently CTRL/G is used giving the value 7. The terminating character ITERM will then become -1 and is used for checking in the program. Read-fields treats every field twice, both before and after data are read. This is demonstrated in the User control (see section 6.7.6)

The termination of reading data is preferably done by giving the terminating function when the cursor is in the first field of the picture. It is possible to 'break' in any field, but the contents of the record already read-in, will then be lost.

THE SOURCE PROGRAM YSFPER

The program is in certain respects abbreviated. See following section on 'Checking status and error reports'.

The source program YSFPER. See comments in the text.

The source program YSFPER. See comments in the text.

```
NORD 10/100 FORTRAN COMPILER FTN-2090H
 1*         PROGRAM YSFPER
 2*  C      FOR DATA INPUT TO PERSON-RECORDS USING PICTURE
 3*  C      PER :  NO USER CONTROL.
 4*  C      PER-C :USER CONTROL ON FIELDS PERSNAME AND HSALARY.
 5*  C      PER-CX:   ..    ..    ..        SEX AND HSALARY.
 6*
 7*  C      ZMUST IS DEFINED FOR FIELDS PERSNAME AND PERSADDR
 8*  C                                          (FIELDS 3 AND 4).
 9*  C      PICTURE FILE: PIC-FILE:OBJ
10*
11*         COMMON/PRIVATE/ITERM(128),IPRIV(1920)
12*         COMMON/PUBLIC/IPUBL(6)
13*         INTEGER PNA(15),FIA(50),FNA(50)
14*         INTEGER REC(50),DEIA(50),FILNO
15*         CHARACTER PFNS*16,PNAS*16,OUTFIL*16
16*         CHARACTER STR1*11,STR2*57,STR3*18
17*         CHARACTER BDATE*6,BNO*5,PERSNAME*26,PERSADDR*30
18*      -           ,SEX*1,DEPARTM*6,PRES*12
19*         EQUIVALENCE(REC(1),STR1),(REC(7),STR2),(REC(36),ISAL)
20*      -           ,(REC(37),STR3)
21*         EQUIVALENCE(REC(1),BDATE),(REC(4),BNO),(REC(7),PERSNAME)
22*      -           ,(REG(20),PERSADDR),(REC(37),DEPARTM)
23*
24*         DATA ITERM/1,5,7B,3,20000B,0,0,0,0,1920,118*0/
25*         DATA IPRIV/0,1920,8,0,0,1915*0/
26*         DATA FILNO/99/,OUTFIL/'PER:DATA'/
27*
28*
29*         WRITE(1,*)'PROGRAM YSFPER FOR INPUT TO PERSON REGISTER'
30*
```

```
C      *** TANDBERG TERMINAL IS ASSUMED HERE ***
       ITERM(2)=3

C      ** PICTURE FILE AND PICTURE ARE GIVEN FROM TERMINAL **
       WRITE(1,20)
 20    FORMAT(/,'$GIVE PICTURE FILE NAME AND TYPE: ')
       READ(1,*)PFNS
       WRITE(1,30)
 30    FORMAT(/,'$GIVE PICTURE NAME: ')
       READ(1,*)PNAS

C      *** OPEN LINE-PRINTER AND OUTPUT FILE FOR PRINT ***
       OPEN(5,FILE='L-P',STATUS='OLD',ACCESS='W')
       WRITE(5,*)' PERSON REGISTER - NEW RECORDS'
       OPEN(FILNO,FILE=OUTFIL,STATUS='UNKNOWN'
     -      ,ACCESS='WA',RECL=45)

C      *** START BY CLEARING THE SCREEN! ***
 35    CALL CLSCR(0,1,23,1,IST)

C      *** THE TWO FIRST 'IMPORTANT' CALLS:
C                     GTPIC AND GTFDN ***
 40    CALL GTPIC(PFNS,1,PNAS,PNA,IST)

       J=1
       IPN=PNA(J)
       NFIND=1
       FIA(1)=0
 50    CALL GTFDN(IPN,NFIND,FIA(1),FNA,NOF,IST)

C      *** WRITE THE PICTURE TO THE SCREEN! ***
 60    CALL WRPTD(IPN,IST)

C      *** TWO FIELDS ARE SET 'MUSTED'! ***
       IRCOD=0
       NOMUST=2
       NSTART=3
       CALL ZMUST(IPN,NOMUST,NSTART,FNA,IST)

C      *** FIELDS ARE CLEARED BEFORE ENTERING NEW DATA ***
 65    CALL CFLDS(IPN,NOF,FNA,IST)

C      *** THE RECORD BUFFER IS CLEARED BETWEEN INPUTS ***
       CALL CLBUF(IPN,NOF,FNA,REC,DEIA,IST)

C      *** THE READ-BITS ARE REMOVED FOR ALL FIELDS ***
       CALL RREAD(IPN,NOF,ISTART,FNA,IST)

C      *** THE CALL FOR ENTERING DATA BY THE TERMINAL ***
 70    CALL RFLDS(IRCOD,IPN,NOF,FNA,REC
     - ,DEIA,NOFR,ITERCHA,IST)

C      *** CTRL/G AT TERMINAL MEANS ITERCHA=-1! ***
       IF(ITERCHA.EQ.-1)GO TO 80

```

```
C        *** WRITE SELECTED FIELDS TO LINE-PRINTER ***
         WRITE(5,75)BDATE,BNO,PERSNAME,PERSADDR,DEPARTM
  75     FORMAT(1H ,A6,X,A5,2X,A26,X,A30,X,A6)
         SAL=ISAL/100

C        *** WRITE STRING FIELDS TO DISK FILE ***
         WRITE(FILNO,76)STR1,STR2,SAL,STR3
  76     FORMAT(A11,A57,F6.2,A18)
         GO TO 65
  80     CALL WMSGE('PROGRAM YSFPER FOR PERSON REG.TERMINATED')
         CALL ZBELL
         STOP
         END
```

## CHECKING STATUS AND ERROR REPORTS

In order not to overload the program example and to emphasize the most important points, the status checking after each call which is a 'must' in normal programs, is omitted. The error procedure for taking care of errors in opening and closing files etc, which should be handled separately, is also left out. To make the example complete, the elements mentioned are given below:

```
*** STANDARD ERROR CHECK EFTER EACH CALL ***

IF(IST.NE.0)THEN
CALL ERROR(´XXXXX´,IST)
GO TO 800
ENDIF
```

here 'XXXXX' should be exchanged by the actual call name!

```
        *** ERRORS IN I-O PROCEDURES ***

800     CALL WMSGE('SUBROUTINE CALL IN ERROR')
        GO TO 850
810     CALL WMSGE('ERROR IN OPEN L-P')
        GO TO 850
820     CALL WMSGE('ERROR IN OPEN OUTFIL')
        GO TO 850
830     CALL WMSGE('ERROR IN WRITE TO L-P')
        GO TO 850
840     CALL WMSGE('ERROR IN WRITE TO OUTFIL')
        GO TO 850
850     CALL WMSGE('PROCESSING TERMINATED')
        CALL ZBELL


        *** ERROR PROCEDURE FOR SUBROUTINE CALLS ***

        SUBROUTINE ERROR(ETXT,IST)
        CHARACTER ETXT*5
        COMMON/PRIVATE/ITERM(10)

        WRITE(1,900)ETXT,IST,ITERM(7)

900     FORMAT(5X,A5,'  IST= ',I5,'   ITERM(7)= ',I5)
        IST=0
        RETURN
        END
```

```
REC        1      4    7         20       35   36    37       40
           1      1    1         1        1    1     1        1
           ------------------------------------------------------
           BDATE  BNO  PERSNAME  PERSADDR SEX  ISAL  DEPARTM  PRES
no of      ------------------------------------------------------ sum
bytes:      6      5      26        30     1    2      6       12

words:      3      3      13        15     1    1      3        6    45

              STR1            STR2             ISAL    STR3
           x---------x-------------------------x-----x------------x
```

*Figure 6—14: Lay-out for record PER. All fields are character strings except ISAL which is integer. Fields with odd no of bytes are rounded up to hold full words.*

### 6.7.5.3. Program Example 2: ZSFPER (F)

We have already mentioned that the data record buffer with the parameter name REC remains unchanged until it is erased or overwritten by new data. If a few fields hold the same contents in several consecutive records, we can utilize this feature for copying the fields from one record to the next one, and thus don't have to enter the same data again and again. In order to make this work, we have to consider a few of the available commands (calls).

DESCRIPTION OF THE PROGRAM

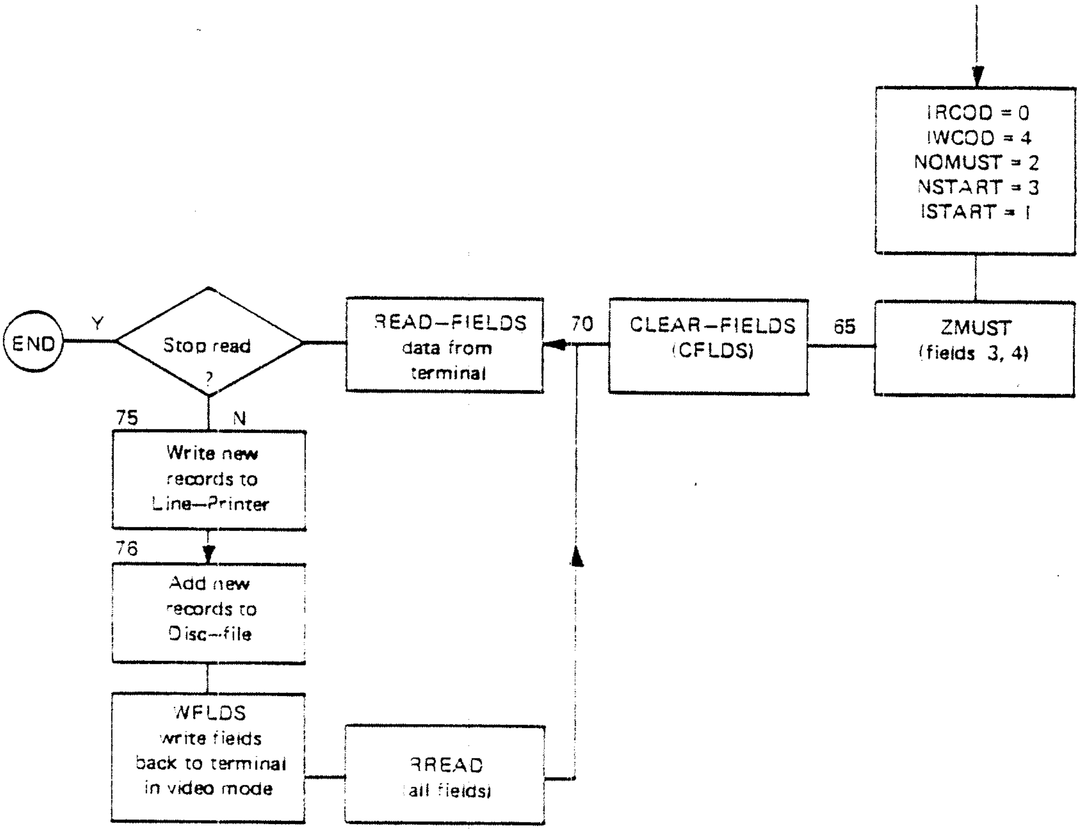


*Figure 6—15: Flow-chart for the last part of program ZSFPER.*

The principle steps of the program are shown in fig 6-15. The first part of the program is identical to YSFPER which is explained above, and we start at the setting of parameters before the ZMUST-call. The new parameter here is IWCOD corresponding to the call WFLDS.

As before the fields no 3 and 4 are 'musted'. Consequently, the call RREAD is used in the loop. The reasons for this is explained in connection with the program YSFPER above.

The fields are initially cleared (outside the loop), and RFLDS is called upon to enter data at the terminal. When the last field in the picture is completed or 'CR' is given, the parameter ITER=0, and the program proceeds with the next step which here is a check on ITER. If we want to terminate reading, CTRL/G is done, giving ITER the value -1. In this program it might be convenient to stop after completing each record to see if it's OK. This can be done by the statement

```
IF(ITER.EQ.0) PAUSE
```

The program will then halt untill 'CR' is given once more. It is also possible by choosing special control keys to conduct the further processing. For example

```
CTRL/L makes ITERCHA=7
CTRL/S   ..   ITERCHA=8 etc. See section 3.6.4.3.
```

The contents of the record is now written out to L-P and disc file, and then the record is written back to the terminal in inverse video display mode, because IWCOD=4 (See section 3.6.4.8).

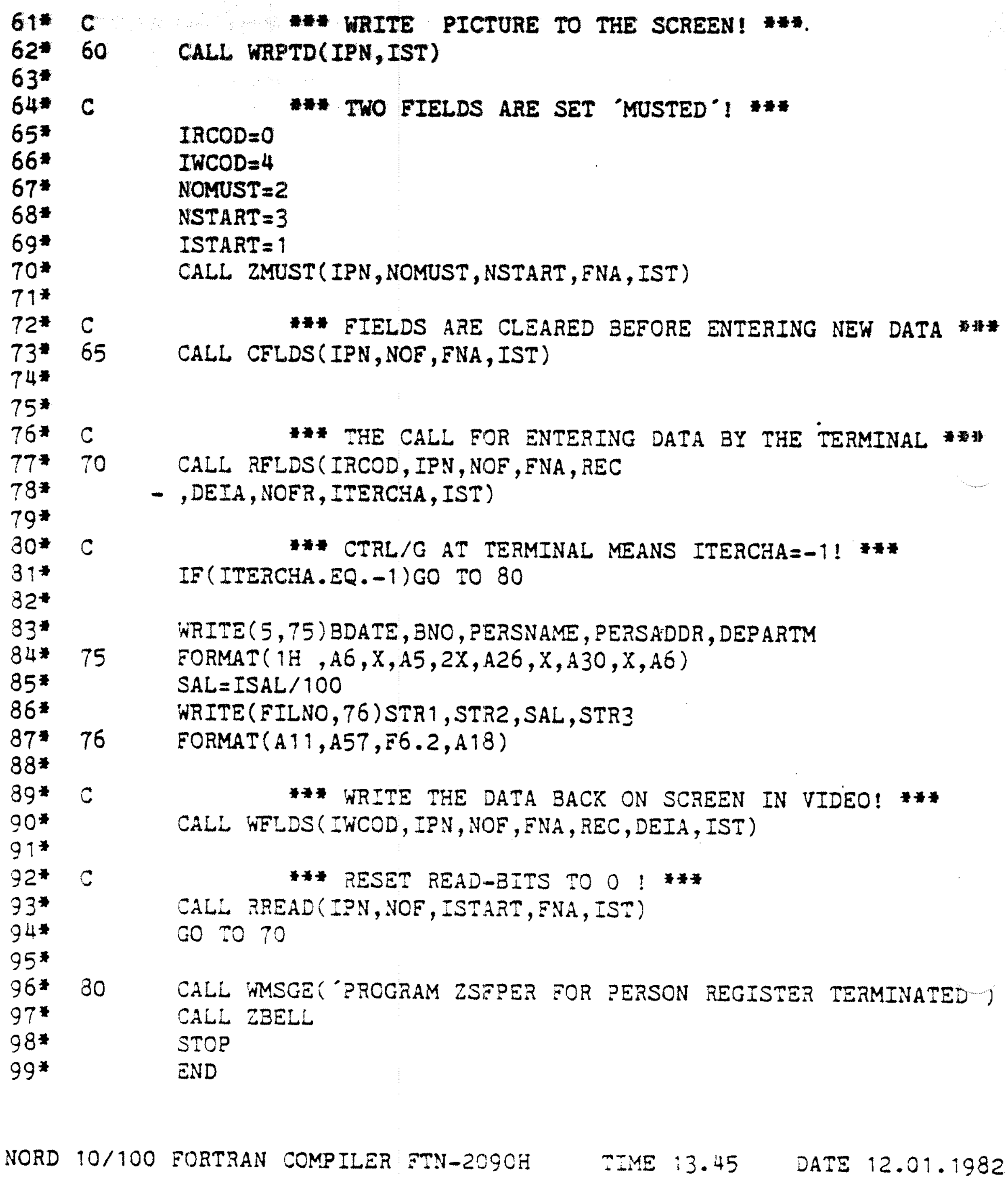
We can now copy old values over to the new record simply by using the down arrow key, except for the fields no 3 and 4 which are 'musted', and for which input must be given. Of course input may be given to any desired field. Because the old values are displayed in inverse video mode and the new input is in normal mode, it is easy to see which fields have been copied, and which ones that have got new input.

In this case it is of course no reason to clear the fields between each record, because data are written back to the screen fields.

The comments given for program YSFPER as to status- checking and error reports, apply here as well.

That part of the source program ZSFPER which corresponds to the flow-chart fig. 6-15 is given below.

The last part of the program ZSFPER corresponding to flow-chart fig. 6-15. The first part is identical to YSFPER.



```
C             *** WRITE  PICTURE TO THE SCREEN! ***.
60      CALL WRPTD(IPN,IST)

C             *** TWO FIELDS ARE SET 'MUSTED'! ***
        IRCOD=0
        IWCOD=4
        NOMUST=2
        NSTART=3
        ISTART=1
        CALL ZMUST(IPN,NOMUST,NSTART,FNA,IST)

C             *** FIELDS ARE CLEARED BEFORE ENTERING NEW DATA ***
65      CALL CFLDS(IPN,NOF,FNA,IST)


C             *** THE CALL FOR ENTERING DATA BY THE TERMINAL ***
70      CALL RFLDS(IRCOD,IPN,NOF,FNA,REC
      - ,DEIA,NOFR,ITERCHA,IST)

C             *** CTRL/G AT TERMINAL MEANS ITERCHA=-1! ***
        IF(ITERCHA.EQ.-1)GO TO 80

        WRITE(5,75)BDATE,BNO,PERSNAME,PERSADDR,DEPARTM
75      FORMAT(1H ,A6,X,A5,2X,A26,X,A30,X,A6)
        SAL=ISAL/100
        WRITE(FILNO,76)STR1,STR2,SAL,STR3
76      FORMAT(A11,A57,F6.2,A18)

C             *** WRITE THE DATA BACK ON SCREEN IN VIDEO! ***
        CALL WFLDS(IWCOD,IPN,NOF,FNA,REC,DEIA,IST)

C             *** RESET READ-BITS TO 0 ! ***
        CALL RREAD(IPN,NOF,ISTART,FNA,IST)
        GO TO 70

80      CALL WMSGE('PROGRAM ZSFPER FOR PERSON REGISTER TERMINATED')
        CALL ZBELL
        STOP
        END


NORD 10/100 FORTRAN COMPILER FTN-2090H      TIME 13.45    DATE 12.01.1982
```

Note: Copying fields from record to record can now be achieved by the special System user control. It works in a different way. See section 6.7.7.

### 6.7.5.4 Program Example 3: YSCPER (C)

This is a COBOL program doing exactly the same as the Fortran program in example 1, and the comments given for that program also apply here. Comments are also given in the source program itself. The principle steps of the program is shown in fig. 6—13. The label numbers correspond to the Fortran program and are not valid here.

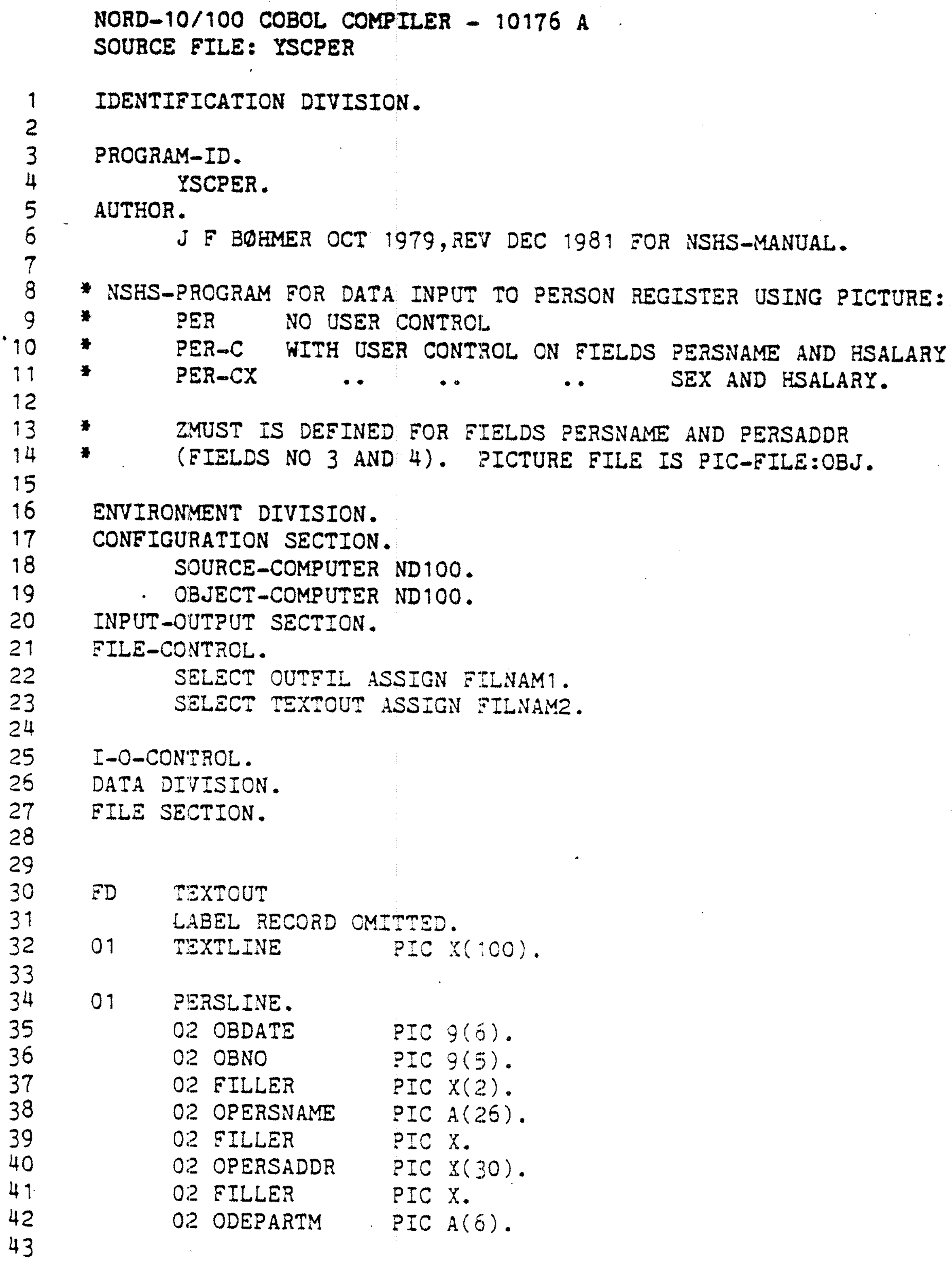
The *common area* in Fortran programs which are used for communicating data between programs and subroutines does not exist in Cobol. In ND systems, the common area is written in the form of a *data block* and compiled with the Fortran compiler. This data block is then loaded before the Cobol program and together with the libraries before dumping. This latter part of the procedure is described in section 6.8.

Because the data block is the first necessary step in the program procedure, we start by giving it here. The data block corresponds directly to the data defined in the common section in the Fortran program:

```
NORD 10/100 FORTRAN COMPILER FTN-2090H

1*          BLOCK DATA
2*          COMMON/PRIVATE/ITERM(128),IPRIV(1920)
3*          COMMON/PUBLIC/IPUBL(6)
4*          DATA ITERM/1,3,100007B,3,20000B,0,0,0,0,1920,118*0/
5*          DATA IPRIV/0,1920,8,0,0,1915*0/
6*          END
```

By means of this procedure, a common area is reserved, and the data are given initial values. However, these values can be changed by using the function (call) COSCR1 as demonstrated in the Cobol program YSCPER below. The letters SC here mean Screen-Cobol. In Cobol, numbers must always be given on decimal form in the value clause. For example, the value 20000B (octal) corresponds to 8192 (decimal).

In the same way as for the Fortran program, the Cobol program YSCPER has been stripped for error checking and error procedures. However, examples of these parts are given after the source program.



```
 NORD-10/100 COBOL COMPILER - 10176 A
 SOURCE FILE: YSCPER

 IDENTIFICATION DIVISION.

 PROGRAM-ID.
       YSCPER.
 AUTHOR.
       J F BØHMER OCT 1979,REV DEC 1981 FOR NSHS-MANUAL.

* NSHS-PROGRAM FOR DATA INPUT TO PERSON REGISTER USING PICTURE:
*      PER     NO USER CONTROL
*      PER-C   WITH USER CONTROL ON FIELDS PERSNAME AND HSALARY
*      PER-CX      ..     ..       ..      SEX AND HSALARY.

*      ZMUST IS DEFINED FOR FIELDS PERSNAME AND PERSADDR
*      (FIELDS NO 3 AND 4).  PICTURE FILE IS PIC-FILE:OBJ.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
       SOURCE-COMPUTER ND100.
       OBJECT-COMPUTER ND100.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
       SELECT OUTFIL ASSIGN FILNAM1.
       SELECT TEXTOUT ASSIGN FILNAM2.

 I-O-CONTROL.
 DATA DIVISION.
 FILE SECTION.


 FD    TEXTOUT
       LABEL RECORD OMITTED.
 01    TEXTLINE         PIC X(100).

 01    PERSLINE.
       02 OBDATE        PIC 9(6).
       02 OBNO          PIC 9(5).
       02 FILLER        PIC X(2).
       02 OPERSNAME     PIC A(26).
       02 FILLER        PIC X.
       02 OPERSADDR     PIC X(30).
       02 FILLER        PIC X.
       02 ODEPARTM      PIC A(6).

```

```
01      ERR-LINE.
        02 ETEXT1        PIC A(9).
        02 EFUNC         PIC X(10).
        02 ETEXT2        PIC X(7).
        02 ESTAT         PIC 9(4).
        02 ETEXT3        PIC X(9).
        02 ENUMB         PIC 9(4).
        02 ETEXT4        PIC X(11).
        02 ECURS         PIC 9(4).

FD      OUTFIL
        RECORDING MODE IS T.
01      OUTREC.
        02 OSTR1         PIC X(70).
        02 OHSALARY      PIC 9(5).
        02 FILLER        PIC X.
        02 OSTR3         PIC X(18).

WORKING-STORAGE SECTION.
        77 PIC-FI-NAM    PIC X(20).
        77 PIC-NAM-STR   PIC X(8).
        77 FILNAM1       PIC X(8) VALUE ´PER:DATA´.
        77 FILNAM2       PIC X(8) VALUE ´L-P´.
        77 DEVICE        PIC 9.
        77 NO-OF-PIC     COMP       VALUE 1.
        77 NO-OF-FI      COMP       VALUE 0.
        77 FI-NO         COMP       VALUE 1.
        77 PIC-NO        COMP       VALUE 1.
        77 FI-IND        COMP       VALUE 0.
        77 R-CODE        COMP       VALUE 0.
        77 STAT          COMP       VALUE 0.
        77 T-PT          COMP       VALUE 10.
        77 CL-CODE       COMP       VALUE 0.
        77 FI-LI         COMP       VALUE 1.
        77 LA-LI         COMP       VALUE 23.
        77 ST-IX         COMP       VALUE 1.
        77 NO-FI-RE      COMP       VALUE 0.
        77 TER-CHA       COMP       VALUE 0.
        77 CL-FI         COMP       VALUE -1.
        77 ITERM6        COMP       VALUE 0.
        77 ITERM7        COMP       VALUE 0.
        77 MUST-NO       COMP       VALUE 2.
        77 ST-IND        COMP       VALUE 3.

        77 MESS1       PIC X(34) VALUE
                           ´START PROGRAM FOR PERSON REGISTER´.
        77 MESS2       PIC X(36) VALUE
                           ´GIVE TERMINAL TYPE=0,1,3,4,5 OR 6: ´.
        77 MESS3       PIC X(32) VALUE
                           ´GIVE PICTURE FILE NAME AND TYPE:´.
        77 MESS4       PIC X(26) VALUE
                           ´GIVE PICTURE NAME STRING:´.
        77 MESS5       PIC X(26) VALUE
                           ´PROGRAM YSCPER TERMINATED´´´.
        77 TXT1        PIC X(40) VALUE
                         ´*** NEW RECORDS TO PERSON REGISTER ***´.
```

```
 01     PIC-NO-AR       PIC X(16).
 01     FI-IND-AR       PIC X(52).
 01     FI-NO-AR        PIC X(52).
 01     DAT-EL-IX-AR    PIC X(52) VALUE LOW-VALUE.

 01     TERM-ARR.
        02 TERM-VAL     COMP      OCCURS 8.

 01     T-A.
        02 TV1          COMP      VALUE 1.
        02 TV2          COMP      VALUE 3.
        02 TV3          COMP      VALUE -32761.
        02 TV4          COMP      VALUE 3.
        02 TV5          COMP      VALUE 8192.
        02 TV6          COMP      VALUE 0.
        02 TV7          COMP      VALUE 0.
        02 TV8          COMP      VALUE 8.
*       -32761 EQUALS 100007B, 8192 IS 20000B (OCTAL).

 01     TERM-STR REDEFINES T-A.
        02 T-S          PIC X(50).


 01     STR-REC         PIC X(90).

 01     REC REDEFINES STR-REC.
        02 BDATE        PIC 9(6).
        02 BNO          PIC 9(5).
        02 FILLER       PIC X.
        02 PERSNAME     PIC A(26).
        02 PERSADDR     PIC X(30).
        02 SEX          PIC A(1).
        02 FILLER       PIC X.
        02 HSALARY      PIC 9(4) COMP.
        02 DEPARTM      PIC X(6).
        02 PRESERV      PIC X(12).

 01     S-REC REDEFINES STR-REC.
        02 STR1         PIC X(70).
        02 S-HSAL       PIC 9(4) COMP.
        02 STR3         PIC X(18).
PROCEDURE DIVISION.
MAIN-PROGRAM SECTION.
P-START.
        DISPLAY MESS1.

        DISPLAY MESS3.
        ACCEPT PIC-FI-NAM.

        DISPLAY MESS4.
        ACCEPT PIC-NAM-STR.

        OPEN OUTPUT TEXTOUT OUTFIL.
        MOVE TXT1 TO TEXTLINE.
        WRITE TEXTLINE AFTER PAGE.
```

```

       MOVE T-A TO TERM-ARR.

*      WITH COSCR1 VALUES FOR ITERM ARE SET FROM THE
*      PROGRAM INSTEAD OF USING VALUES FROM DATA BLOCK.
       CALL ´COSCR1´ USING TERM-ARR.

       CALL ´CLSCR´ USING CL-CODE FI-LI LA-LI ST-IX STAT.

       CALL ´GTPICC´ USING
         PIC-FI-NAM NO-OF-PIC PIC-NAM-STR PIC-NO-AR STAT.

       CALL ´GTFDN´ USING
                  PIC-NO-AR FI-NO FI-IND FI-NO-AR NO-OF-FI STAT.

       CALL ´WRPTD´ USING PIC-NO STAT.

       CALL ´ZMUST´ USING PIC-NO MUST-NO ST-IND FI-NO-AR STAT.

 NEW-REC.
       CALL ´CFLDS´ USING
                       PIC-NO NO-OF-FI FI-NO-AR STAT.

       CALL ´CLBUF´ USING
                       PIC-NO NO-OF-FI FI-NO-AR
                       STR-REC DAT-EL-IX-AR STAT.

       CALL ´RREAD´ USING PIC-NO NO-OF-FI FI-NO FI-NO-AR STAT.

       CALL ´RFLDS´ USING
                    R-CODE PIC-NO NO-OF-FI FI-NO-AR
                STR-REC DAT-EL-IX-AR NO-FI-RE TER-CHA STAT.

       IF TER-CHA = -1 GO TO FIN.

*      WRITE BDATE,BNO,PERSNAME,PERSADDR,DEPARTM TO L-P:
       MOVE SPACE TO PERSLINE.
       MOVE BDATE TO OBDATE.
       MOVE BNO TO OBNO.
       MOVE PERSNAME TO OPERSNAME.
       MOVE PERSADDR TO OPERSADDR.
       MOVE DEPARTM TO ODEPARTM.
       WRITE PERSLINE AFTER 1.

*      WRITE RECORD TO DISC-FILE:
       INSPECT S-HSAL REPLACING LEADING SPACE BY ´0´.
       MOVE SPACE TO OUTREC.
       MOVE STR1 TO OSTR1.
       MOVE S-HSAL TO OHSALARY.
       MOVE STR3 TO OSTR3.
       WRITE OUTREC.

       GO TO NEW-REC.

 FIN.
```

```
            CLOSE TEXTOUT OUTFIL.

            CALL 'WMSGEC' USING MESS5.
            CALL 'ZBELL'.
            STOP RUN.
```

The experienced Cobol programmer will probably find no particular difficulties in this program.

Observe that a few of the NSHS function names (not all) have got an extra 'C' at the end, for example 'GTPICC', 'WMSGEC' (in Fortran: 'GTPIC', 'WMSGE' etc.). For all parameters standard names have been used. See section 3.6.3.15.

All error-checking and error reports have been skipped in this program, but may look as shown below.

The following check should be carried out after each call. The 'XXXXX' should be exchanged by the actual call name.

```
IF STAT NOT = 0
MOVE 'XXXXX' TO E-FUNC
PERFORM ERR-REP
GO TO FIN.
```

If errors occur in run-time, the parameters ITERM(6) and ITERM(7) are updated with curser position and error code number and can be called by the standard function 'ERRORC'. The error-rep section is usually placed at the end of the program.

```
ERR-REP SECTION.
E-S.
*      FOR ERROR STATE.
       CALL 'ERRORC' USING ITERM7 ITERM6.
       MOVE STAT TO E-STAT.
       MOVE ITERM7 TO E-NUMB.
       MOVE ITERM6 TO E-CURS.
       MOVE E-TEXT TO ERR-LINE.
       WRITE ERR-LINE AFTER 2.
       DISPLAY E-TEXT.
E-E.
```

### 6.7.5.5. Program Example 4: ZSCPER (C)

This COBOL program corresponds to the Fortran program explained in example 2. The first part of the program is identical to the preceeding YSCPER, and only the last part is handled here. The principle steps are shown in fig. 6-15 by flow-chart. Most of the comments given for example 2 are valid here.

The last part of the program ZSCPER corresponding to flow-chart fig.6—15. The first part is identical to YSCPER.

```
             CALL 'WRPTD' USING PIC-NO STAT.

             CALL 'ZMUST' USING PIC-NO MUST-NO ST-IND FI-NO-AR STAT.

             CALL 'CFLDS' USING
                             PIC-NO NO-CF-FI FI-NO-AR STAT.

       NEW-REC.

             CALL 'RFLDS' USING
                          R-CODE PIC-NO NO-CF-FI FI-NO-AR
                      SCR-REC DAT-EL-IX-AR NO-FI-RE TER-CHA STAT.

             IF TER-CHA = -1 GO TO FIN.

      *      WRITE BDATE,BNO,PERSNAME,PERSADDR,DEPARTM TO L-P:
             MOVE SPACE TO PERSLINE.
             MOVE BDATE TO OBDATE.
             MOVE BNO TO OBNO.
             MOVE PERSNAME TO OPERSNAME.
             MOVE PERSADDR TO OPERSADDR.
             MOVE DEPARTM TO ODEPARTM.
             WRITE PERSLINE AFTER 1.

      *      WRITE RECORD TO DISC-FILE:
             INSPECT S-HSAL REPLACING LEADING SPACE BY '0'.
             MOVE SPACE TO OUTREC.
             MOVE STR1 TO OSTR1.
             MOVE S-HSAL TO OHSALARY.
             MOVE STR3 TO OSTR3.
             WRITE OUTREC.

             CALL 'WFLDS' USING
                          W-CODE PIC-NO NO-CF-FI FI-NO-AR
                          SCR-REC DAT-EL-IX-AR STAT.

             CALL 'RREAD' USING PIC-NO NO-CF-FI FI-NO FI-NO-AR STAT.

             GO TO NEW-REC.

       FIN.
             CLOSE TEXTOUT OUTFIL.

             CALL 'WMSGEC' USING MESS5.
             CALL 'ZBELL'.
             STOP RUN.
```

Another COBOL example is given in Appendix H.

## 6.7.6. *User Control*

When a picture is created by means of the SCREEN-DEF program, it is possible to check the contents of the data entered to the fields in different ways. This control is a standard system control inherent in the NSHS system and identified by the edit codes 1—10. See section 2.5.5.

Of course, the standard controls should be used if possible. However, if a very special check is to be carried out, or the value of data fields are being compared, for example, this can be done by a User control. The edit code is then set to 9, and a special user control number is given to that field.

The User control is written as a Fortran subroutine where the first parameter is the user control number.

All controls, even the User control, are initiated from the picture. It is then possible to create a number of pictures which are identical except with respect to the User control. On the picture file PIC-FILE we have three different pictures PER, PER-C and PER-CX which are identical except in respect to User control. PER has no control, while the others have User controls on different fields. See the picture descriptions table 6-3, 6-4 and 6-5.

The standard name of the User control subroutine is

```
UCONT(IUSCOD,.........)
```

with parameters in the paranthesis. The first parameter is the user control number, and one of the first statements in the subroutine will normally be a jump to the right spot based on IUSCOD (branching).
There can only be *one* UCONT, and all the User controls must be placed in this subroutine. Ucont is compiled by the Fortran compiler compatible with the Screen-Library to be used.

The Fortran compiler FTN corresponds to the files SCREEN -LIB or SCREEN-RTL; the compiler called FORTRAN is used in connection with SCREEN-1BANK, SCREEN-2 BANK and SCREEN-1REEN. Observe that the *file name* for UCONT may be any legal name, preferably some name connected to the picture.

User control programs often require manipulation of data on bit and byte level and a very good knowledge both to Fortran and NSHS. It is no beginners' task! For the purpose of User control, a few standard subroutines have been written. A description of these subroutines are given in Appendix F.

In order to demonstrate a few features of User control and how it works, a subroutine with the file name SPER-CONT is written and listed below. But first a brief description of the tasks of the program is given.

## 6.7.6.1 Description of the User Control SPER-CONT (UCONT)

The picture PER has no call to UCONT and is not relevant here. If the picture wanted is given interactively from the terminal, *one* main program and *one* user subroutine will do.

PICTURE PER-C

1) When Read-fields (RFLDS) is called in the program, the fields are checked twice, before and after data are entered. By proper setting of the Status parameter for field no 3 (PERSNAME) we demonstrate this by writing a message to the screen both before and after data are entered in the field. The messages are accompanied by a 'beep' on the terminal (ZBELL).

2) The next control is initiated on field no 6 (HOURS' PAY) named HSAL in the program. The age of the person is found based on BDATE, field no 2, and the persons' salary compared to his age. If a disagreement is found, an error message is displayed and the input is not accepted.

PICTURE PER-CX

1) The first check here is identical with the previous one for picture PER-C, checking salaries against age. Consequently the user control number must be the same in both cases =2.

2) Here a check is carried out on the field SEX. If the data entered is 'M', the following fields are set 'musted', else (if 'F') the following fields are skipped. It appears a little tricky to find the character because it is placed left oriented in the field.

Source program SPER-CONT (User control subroutine).

```
NORD 10/100 FORTRAN COMPILER FTN-2090H
      SUBROUTINE UCONT(IUSCOD,IPN,FINFO,NOF,FNA,REC
     -                         ,DEIA,INDEX,REDBITS,IST)
C               *** FILE NAME FOR UCONT IS SPER-CONT

      INTEGER FINFO(1),FNA(1),REC(1),DEIA(1),REDBITS(1)
      INTEGER IREC(50),TIMARR(7),DATO(3)
      INTEGER ISHFT
      REAL HSAL
      CHARACTER SCH*1
      EQUIVALENCE(IREC(36),ILONN)

C     *** USER CONTROL IS CALLED FROM THE PICTURES PER-C AND
C     *** PER-CX AS FOLLOWS. PICTURE PER USES NO CONTROL.
C     *** PICTURE FILE: PIC-FILE:OBJ.

C     *** PICTURE PER-C  FOR CONTROL 1+2
C     *** PICTURE PER-CX FOR CONTROL 2+3

C     *** 1.BEFORE READING IN DATA TO PERSNAME,A MESSAGE
C     ***   IS WRITTEN TO THE SCREEN,AND AFTER TERMINATING
C     ***   THE FIELD ANOTHER MESSAGE IS DISPLAYED (LINE 24).

C     *** 2.AGE OF PERSON (IAGE) IS CALCULATED BY COMPARING
C     ***   BDATE  REC(1)-(3) WITH THE DATE TO-DAY.

C     ***   INPUT TO FIELD HSALARY IS CHECKED AGAINST:
C     ***    IAGE<18      10.<=HSAL<25.
C     ***    IAGE>=67     25.<=HSAL<50.
C     ***    18<=IAGE<67  25.<=HSAL<=250.

C     ***   IF CHECK NOT OK, ERROR REPORT ON SCR LINE 24.

C     *** 3.IF SEX='M' FOLLOWING 3 FIELDS ARE SET MUSTED,
C     ***   ELSE THE SAME FIELDS ARE LOCKED.

C              *** GO TO ACTUAL CONTROL:
      GO TO(10,50,80)IUSCOD

C              *** START CONTROL 1.
10    IF(IST.EQ.-1)THEN
      CALL WMSGE('READ FIELD PERSNAME')
      CALL ZBELL
      GO TO 300
      ELSE
      CALL WMSGE('FIELD PERSNAME READ')
      CALL ZBELL
      GO TO 100
      ENDIF

C              *** START CONTROL 2.
C              *** IF NOTHING READ TO FIELD,RETURN:
50    IF(IST.EQ.-1)GO TO 300

```

```
         DO FOR K=1,50
         IREC(K)=REC(K)
         ENDDO
         READ(IREC,60)(DATO(I),I=1,3)
   60    FORMAT(3I2)
   C          *** COMPUTE IYEAR,IMON AND IDAY. BDATE IN REC(1)-(3)
         CALL CLOCK(TIMARR)
         IYEAR=TIMARR(7)-1900-DATO(3)
         IMON=TIMARR(6)-DATO(2)
         IDAY=TIMARR(5)-DATO(1)
         IF(IMON.GT.0)GO TO 70
         IF(IMON.LT.0)THEN
         IYEAR=IYEAR-1
         GO TO 70
         ENDIF
         IF(IDAY.GE.0)GO TO 70
         IYEAR=IYEAR-1
   C            *** CHECK HSALARY AGAINST IYEAR
   70    HSAL=FLOAT(ILONN/100)
         IF(IYEAR.LT.18)THEN
            IF((HSAL.LT.10).OR.(HSAL.GE.25))THEN
            CALL WMSGE('FOR AGE<18   10<=HSAL<25')
            CALL ZBELL
            GO TO 300
            ENDIF
            GO TO 100
         ENDIF
         IF(IYEAR.GE.67)THEN
            IF((HSAL.LT.25).OR.(HSAL.GE.50))THEN
            CALL WMSGE('FOR AGE>=67    25<=HSAL<50')
            CALL ZBELL
            GO TO 300
            ENDIF
            GO TO 100
         ENDIF
         IF((HSAL.LT.25).OR.(HSAL.GT.250))THEN
            CALL WMSGE('18=<AGE<67    25<=HSAL<=250')
            CALL ZBELL
            GO TO 300
         ENDIF
         GO TO 100

   C                    *** START CONTROL 3.
   C                    *** IF NOTHING READ TO FIELD,RETURN
   80    IF(IST.EQ.-1)GO TO 300
   C            *** CHECK IF 'M' OR 'F'(IN LEFT BYTE OF WORD 35)
   C            *** NOW RIGHT BYTE IS ERASED AND
   C            *** LEFT BYTE MOVED TO THE RIGHT.
         ICHAR=REC(DEIA(INDEX)).AND.177400B
         ICHAR=ISHFT(ICHAR,-8)
         SCH=CHAR(ICHAR)
         NOMUST=3
         NOLOCK=3
         NSTART=6
```

```
C                       *** RESET MUST AND LOCK.
       CALL RMUST(IPN,NOMUST,NSTART,FNA,IST)
       CALL RLOCK(IPN,NOMUST,NSTART,FNA,IST)

       IF(SCH.EQ.'M')THEN
          CALL ZMUST(IPN,NOMUST,NSTART,FNA,IST)
       ELSE
          CALL ZLOCK(IPN,NOLOCK,NSTART,FNA,IST)
       ENDIF
C               *** CHECK OK.
100    IST=0
       RETURN
C               *** SKIP TO NEXT FIELD.
200    IST=1
       RETURN
C               *** CHECK WRONG,TRY AGAIN
300    IST=-1
       RETURN
       END
```

COMMENTS TO THE SUBROUTINE

Control 1: At the arrival to field no 3, no data has been entered, giving status=-1, which is the same as for 'check wrong'. The jumps to the labels 100, 200 and 300 are due to 'programming style'. The point here is to show that each field is checked twice during a Read-fields call, which is clearly demonstrated.

Control 2: This control contains first the call clock(.....) which puts the computer's time into the array TIMARR. The following algorithme extracts the age of the person which is used to compare salary against age.

Control 3: By means of REC(DEIA(INDEX)) we get the address to the element SEX without having to count words in the record. From fig.6-14 we see that SEX is found in REC(35), but that information is here obtained from the system. The field occupies only one byte, which in this case is the left one, because the field contains only one character. This means that we have to erase the right byte (containing a pad character) by means of the .AND. option (see section 6.7.4).

Next step is to move the left byte to the right position by ISHFT(...) and finally get the data on string form by means of CHAR(....).

All Fortran features used here are found in the ND manuals ,for example NORD-10/100 FORTRAN System Reference manual, ND-6.074.03

## 6.7.7 *System User Control (Screen-Ucont)*

The program examples (2) and (4) above demonstrated how programs can be written to copy certain fields from one record to another, if they hold the same data.

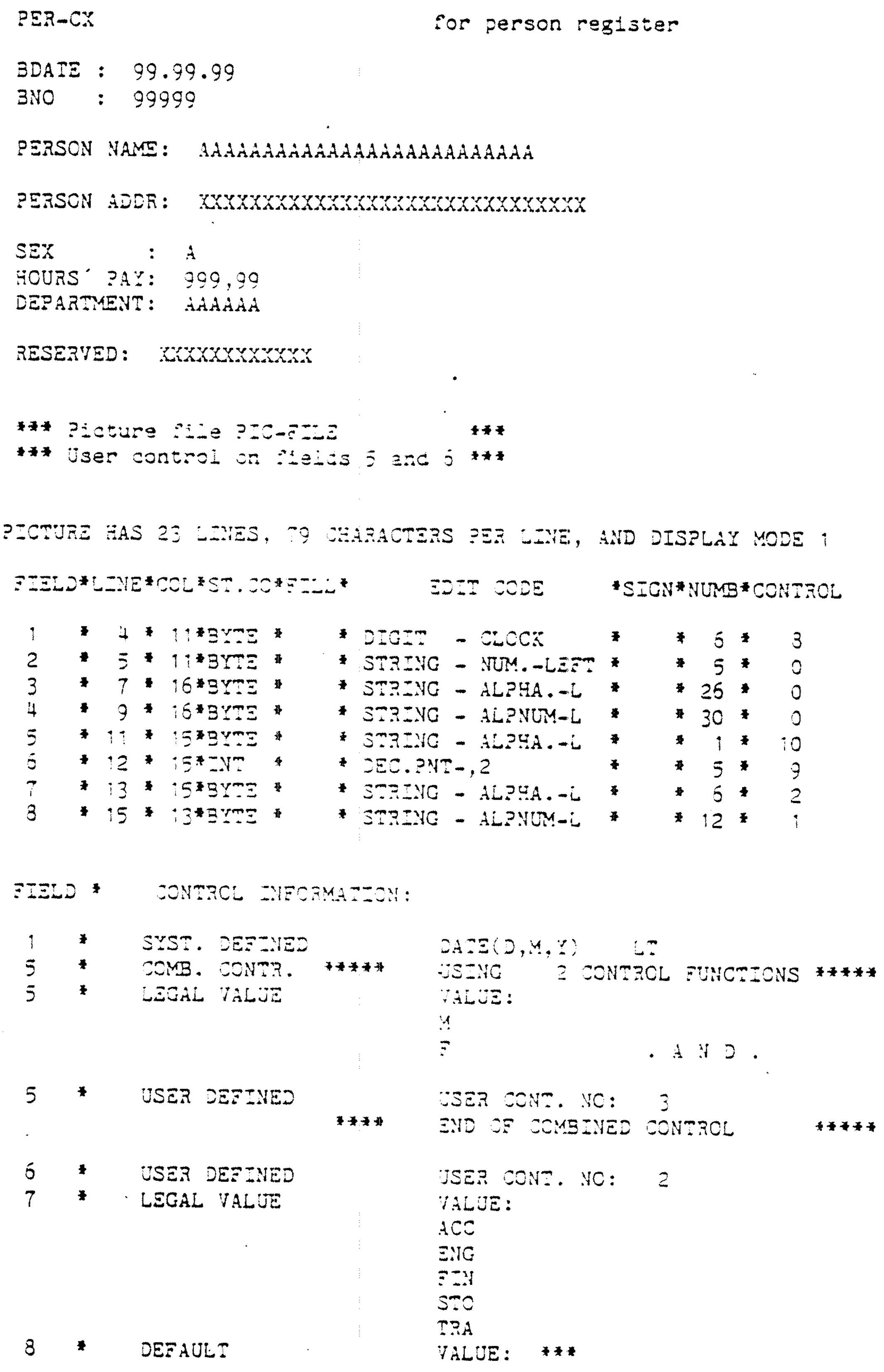
Almost the same effect is achieved by the System User Control called Screen-Ucont. It is on symb-form and may thus be edited before compiling. The copying effect is obtained by entering data to the fields, and then locking the fields to be copied. Locked fields are jumped over and treated like leading text.

The process is activated by means of an extra final field, defined with one character and User control no=1. The following text is taken from ND's program description SCREEN-UCONT.

The reader may compare the program examples (2) or (4) above with the Screen-Ucont procedure. They work completely different!

```
PER-C                                for person register

BDATE :  99.99.99
BNO   :  99999

PERSON NAME:  AAAAAAAAAAAAAAAAAAAAAAAAAA

PERSON ADDR:  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SEX        :  A
HOURS´ PAY:  999,99
DEPARTMENT:  AAAAAA

RESERVED:  XXXXXXXXXXXX


*** Picture file PIC-FILE          ***
*** User control on fields 3 and 6 ***


PICTURE HAS 23 LINES, 79 CHARACTERS PER LINE, AND DISPLAY MODE 1

FIELD*LINE*COL*ST.CO*FILL*      EDIT CODE      *SIGN*NUMB*CONTROL

  1  *  4 * 11*BYTE *   * DIGIT  - CLOCK     *    *  6 *    8
  2  *  5 * 11*BYTE *   * STRING - NUM.-LEFT *    *  5 *    0
  3  *  7 * 16*BYTE *   * STRING - ALPHA.-L  *    * 26 *    9
  4  *  9 * 16*BYTE *   * STRING - ALPNUM-L  *    * 30 *    0
  5  * 11 * 15*BYTE *   * STRING - ALPHA.-L  *    *  1 *    2
  6  * 12 * 15*INT  *   * DEC.PNT-,2         *    *  5 *    9
  7  * 13 * 15*BYTE *   * STRING - ALPHA.-L  *    *  6 *    2
  8  * 15 * 13*BYTE *   * STRING - ALPNUM-L  *    * 12 *    1


FIELD *    CONTROL INFORMATION:

  1   *    SYST. DEFINED          DATE(D,M,Y)     LT
  3   *    USER DEFINED           USER CONT. NO:   1
  5   *    LEGAL VALUE            VALUE:
                                  M
                                  F
  6   *    USER DEFINED           USER CONT. NO:   2
  7   *    LEGAL VALUE            VALUE:
                                  ACC
                                  ENG
                                  FIN
                                  STO
                                  TRA
  8   *    DEFAULT                VALUE:  ***
```

*Table 6-4: Picture description for PER-C.*



```
PER-CX                              for person register

BDATE :  99.99.99
BNO   :  99999

PERSON NAME:  AAAAAAAAAAAAAAAAAAAAAAAAAA

PERSON ADDR:  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SEX        :  A
HOURS´ PAY:  999,99
DEPARTMENT:  AAAAAA

RESERVED:  XXXXXXXXXXXX


*** Picture file PIC-FILE          ***
*** User control on fields 5 and 6 ***


PICTURE HAS 23 LINES, 79 CHARACTERS PER LINE, AND DISPLAY MODE 1

FIELD*LINE*COL*ST.CO*FILL*       EDIT CODE       *SIGN*NUMB*CONTROL

  1    *  4 * 11*BYTE *     * DIGIT  - CLOCK     *     *  6 *    8
  2    *  5 * 11*BYTE *     * STRING - NUM.-LEFT *     *  5 *    0
  3    *  7 * 16*BYTE *     * STRING - ALPHA.-L  *     * 26 *    0
  4    *  9 * 16*BYTE *     * STRING - ALPNUM-L  *     * 30 *    0
  5    * 11 * 15*BYTE *     * STRING - ALPHA.-L  *     *  1 *   10
  6    * 12 * 15*INT  *     * DEC.PNT-,2         *     *  5 *    9
  7    * 13 * 15*BYTE *     * STRING - ALPHA.-L  *     *  6 *    2
  8    * 15 * 13*BYTE *     * STRING - ALPNUM-L  *     * 12 *    1


FIELD *    CONTROL INFORMATION:

  1    *     SYST. DEFINED              DATE(D,M,Y)    LT
  5    *     COMB. CONTR.   *****       USING    2 CONTROL FUNCTIONS *****
  5    *     LEGAL VALUE                VALUE:
                                        M
                                        F                 . A N D .

  5    *     USER DEFINED               USER CONT. NO:   3
                             ****       END OF COMBINED CONTROL       *****

  6    *     USER DEFINED               USER CONT. NO:   2
  7    *     LEGAL VALUE                VALUE:
                                        ACC
                                        ENG
                                        FIN
                                        STO
                                        TRA
  8    *     DEFAULT                    VALUE:  ***
```

*Table 6-5: Picture description for PER-CX.*

## 6.8 *COMPILING, LOADING AND RUNNING PROGRAMS*

The procedure of compiling, loading and executing programs is no special NSHS task and might be omitted here.

However, in connection with NSHS a few details must be observed, as explained in the following.

The source program in the form of text strings must be *compiled* to obtain the object program which can be used by the computer.

A program may be either a RT-program, a reentrant program or a back-ground program. The type of program depends both on the program source and how it is used in the computer environment.

The program examples above are all simple back-ground programs, and the procedures for compiling etc. demonstrated below are valid for this kind of programs.

### 6.8.1 *Compiling, Loading and Running a Fortran Program*

COMPILING

In connection with NSHS, two different versions of the Fortran compiler, can be used. See note in (6.7.6).The version name can be seen by the compiling outprint for the program example below.

The compiling sequence:

```
@ftn
NORD 10/100 FORTRAN COMPILER FTN-2090 H
$comp ysfper, l-p, " ysfper"
100 LINES COMPILED. OCTAL SIZE=1467
CPUTIME USED IS 4.0 SEC.
$ex
@
```

The text in lower case is keyed by the operator, the text in capital letters is the response from the compiler. On the 'comp' line we have the name of the source file, the list file and the object file. If the object file already exists, the quotation marks are omitted. After the compiling we have the two files YSFPER:SYMB and YSFPER:BRF. The latter is used in the loading procedure which is the next step.

## LOADING AND RUNNING THE PROGRAM

The Nord Relocating Loader (NRL) is used for all background programs, both Fortran and Cobol.

The sequence is:

```
@nrl
RELOCATING LOADER LDR-1935H
*size 600
*load ysfper
*x-load sper-cont        (must be loaded before scr-lib)
*load scr-lib
*load ftnlib
FREE 110273-173600
*
*dump " ysfper "
*run or *ex
...........
@
```

The command *size is given to avoid 'table loader overflow'. The program and libraries may be loaded one by one like here, or by one command (line).

The files loaded are the object file, the user control subroutine, the NSHS library and the Fortran library in the given order. The available space at the start of the loading is 0 - 177777B equal to 64k words. The programs successively occupy space, the common area from the top, the programs from the base. The available space for processing is here the interval 110273 to 173600 octal.

If the program has been successfully loaded, it may be dumped as shown. We then get the program on the form YSFPER:PROG, and it is then possible to call the program simply by keying @YSFPER. After loading, the program may be executed by the command *run, or we return to Sintran by *ex.

## 6.8.2 *Compiling, Loading and Running a Cobol Program*

The principles of these procedures are the same as for the Fortran programs. However, in this case a few more details must be considered, as shown in the following.

COMPILING

Both old and new compilers may be used. If an old compiler is used, the FD clause for defining symbolic files (disk files), for ex in the program example (3), YSCPER, line 55: 'RECORDING MODE IS T.' must be replaced by 'BLOCK CONTAINS 0 RECORDS.'

The sequence is:

```
@Cobol
NORD-10/100 COBOL COMPILER - 10176A
*comp yscper, l-p, ''yscper''

*** NO ERROR MESSAGES ***
*ex
```

Letters in lower case are keyed by the operator, capital letters are the response from the compiler. On the 'comp' line we have the source file, the list file and the object file. If the object file already exists, the quotation marks are omitted. After the compiling, we have the two files YSCPER:SYMB and YSCPER:BRF, the latter is the object file used for loading, which is the next step.

LOADING AND RUNNING THE PROGRAM

The first file to be loaded is the Data block for allocating the common area (see section 6.7.5.4 for program ex.3).

The loading sequence is:

```
@nrl
RELOCATING LOADER LDR-1935H
*size 600
*load sper-block
*load yscper
*x-load sper-cont        (Must be loaded before screen-lib)
*load scr-lib
*load ftnlib
*load cob-lib
FREE 126370-173600
*dump ''yscper''
*run or *ex
@
```

The command 'size' is given to prevent 'loader table overflow'.

The files in the loading sequence are:

sper-block The data block allocating the common area (written and compiled with Fortran).

yscper The main cobol program.

sper-cont The user control program (Fortran).

scr-lib The screen library for NSHS.

ftnlib The old Fortran library.

cob-lib The new Cobol library (The old one was called 'runcob' and may be used. See comments above).

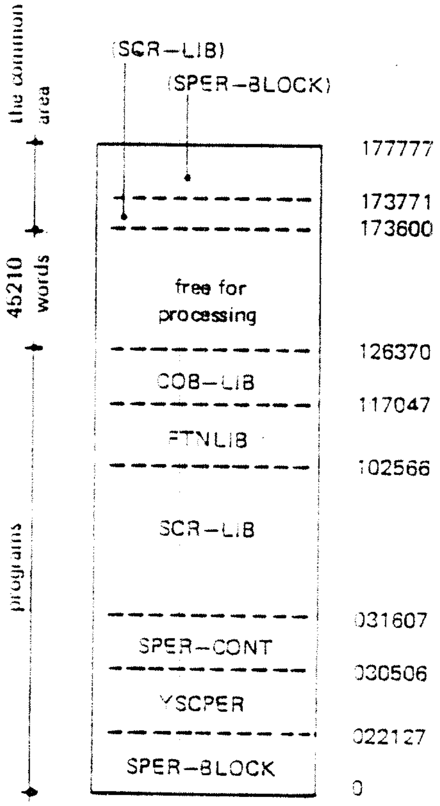


*Figure 6—16: The space occupied in the virtual memory by programs and common.*
*The free space is 173600-126370=45210B words.*
*The figure is not in scale.*

If the loading is successful, the message FREE..... is displayed. If some library or data block is forgotten, the message ENTRIES UNDEFINED ....... is displayed, giving the names of the missing routines. Sometimes we want a listing of the subroutines and their start points, and then use the command *entries-defined.

In the loading procedure above, all the programs are loaded separately one by one. It is then easy to see from the message FREE.... how much space is occupied by each program. However, all the programs may be loaded by one command (line). The virtual user space which is 64k words is in this case partly occupied by programs and common as shown in fig.6-16.

When the programs have been successfully loaded, they may be 'dumped' as shown. The program may then later be initiated from Sintran by @YSCPER.

Obs that all input to the NSHS programs must be given with *capital* letters. For example, if the program YSCPER is called, we are asked about 'file name and type:' The answer then may be 'PIC-FIL:OBJ' (or abbreviated P-F:O). If not, the terminal will probably be 'hanging' or an error message issued.

# APPENDIX A

# EDIT CODES AND FORMATS

When defining a field, the user may specify an editing option. NSHS contains a table of 29 standard formats, stored in the labelled common area with the name FORMAT. A format consists of a series of words in the area FORMAT. When these words are read in ascending order (i.e., IFORMAT (N), IFORMAT ( N + 1), IFORMAT ( N + 2) ...) they describe the format of the characters to be printed from *RIGHT→ LEFT*. The contents of these words are interpreted as follows:

| *Value:* | *Meaning:* |
|---|---|
| 1 | Print the sign (if any), space or — |
| 2 | Print next character |
| 3 | Print thousands separator for decimal fields |
| 4 | Print decimal position for decimal fields |
| > 31 | Print the ASCII character represented by value. That is, a format character. |
| < 0 | Loop in the format |

An example will illustrate how it works. Consider edit code 2. *(See Section 2.5.4.2)*. This edit code will print a negative 10 significant digit number as follows:

12.345.678,90—

and the format string for this is:

1,2,2,4,2,2,2,3,—4,

3 and 4 are the codes for the thousands separator character and the decimal position character, respectively.

The two characters to be used for formatting decimal fields are stored in the 41st word of a labelled common area called PICDEF. The word normally contains the ASCII code for (, .). In England this should be patched to (. ,).

Suppose, for example, you wished to print a character string read into a byte field as follows:

C*CCCC*CC/ /CCCC*

**Here, C represents a character read in and the * and/represent format characters. The format string for this would be:**

**45,2,2,2,2,47,47,2,2,45,2,2,—5**

**The correct position in the format array is determined by the edit code, and a pointer array called FPOINT. This pointer array contains one word for each edit code. Each word contains:**

**Left byte:** **0 or terminating character for this edit code**

**Right byte:** **index to start of format string in FORMAT**

You may change the contents of both FPOINT and FORMAT, but be aware of the following restrictions.

1. Edit code 0 - 9 must have at least one format character.

2. Format characters cannot be defined to appear left-most in a field (include them in the leading text!).

3. The contents of FPOINT and FORMAT MUST BE IDENTICAL when defining pictures and using the same pictures.

4. FORMAT should not be greater than 256 words.

Both FPOINT and FORMAT for the standard NSHS system are shown below.

```
*)9BEG;)9ENT FPOIN

% FPOINT is the format pointers array with one word for each edit code.
% The contents of each word are as follows:

% Left byte:    terminating character for fields with this edit code (no parity bit
                set), or zero

% Right byte:   index to start of format string in FORMAT array
```

```
SUBR FPOINT
@ICR
FPOINT:; INTEGER ARRAY PPPP:=(
  0 \ 20,0 \ 22,0 \ 24,0\ 26,0 \ 30,0 \ 32,0 \ 34,0 \ 36,
  0 \ 40,0 \ 74,0 \ 42,0\ 45,0 \ 60,0\ 42,0 \ 42,0 \ 42,
  0\ 42,0\ 42,0\ 42,0\ 42,0\ 65,0 \ 65,0 \ 65,0 \ 67,
  0 \ 67,0 \ 67,0 \ 65,0\ 65,0 \ 65) ; @CR
RBUS
*)9END
  COMMON /FORMAT/ IFORMAT(80)
  DATA IFORMAT/
- 2,2,2,2,2,2,2,2,2,4,2,2,2,3,-4,
- 1,-8,1,-11,1,-14,1,-17,1,-20,1,-23,1,-26,1,-29,1,-32,
- 1,2,-1,2,2,2,2,2,56B,2,2,56B,2,-1,1,2,2,56B,-3,
- 2,-1,2,2,2,57B,-4,1,-50,19*0/
```

Symbolic versions of the above are supplied as part of the NSH system.

*Changes from the previous version of NSHS (ND.10013H)*

Left justified fields are handled different from the previous version of NSHS.

a. Fields of type Integer, Double integer, Fortran I5 and BCD with edit code F, G, H, I or J and fill code =1, i.e. filled with 0, now hold the same value as displayed on the screen.

An example:

| Input | Value on the screen | Old data element | New data element |
|---|---|---|---|
| 1cr | 1000 | 0001 | 1000 |

b. Fields of type Byte with edit code F, fill code 0, i.e. unused positions filled with blanks, are now right justified in the data element with leading unused positions filled with blanks. Before, these fields were left justified in the data element, with the unused posisions to the *right* filled with blanks.

Two examples:

| Input | Value on the screen | Old data element | New data element |
|---|---|---|---|
| 1cr | 1bbb | 1bbb | bbb1 |
| 01cr | 1 | **** | bbb1 (not allowed before) |

c. Fields of type Byte with edit codes F, and fill code 1, i.e. unused positions filled with 0, are now right justified in the data element with leading unused positions filled with blanks. It is now permitted to key leading zeroes in such fields. Leading zeroes can be considered as unused positions.

A few examples:

| Input | Value on the screen | Old data element | New data element |
|---|---|---|---|
| 1cr | 1000 | 1000 | 0001 |
| 01cr | 0100 | **** | b100 (not allowed before) |

*Table for byte fields with data elements*

| EDIT<br>CODE | FILL=0<br>(SPACE) | FILL= 1<br>(ZERO) | LEGAL<br>CHAR | INPUT<br>EX | COMMENT |
|---|---|---|---|---|---|
| | | *** NUMERIC FIELDS *** | | | |
| 0 - 9 | bbb99,00<br>bbb9900 | 00.099,00<br>bbb9900 | '0-9','+-,.' | 9900 | Edit code 2 used here. If sign is defined one + or - is allowed |
| A, | bbb9900 | 0009900 | '0-9','+-' | 9900 | If sign is defined, one + or - is allowed. |
| B | bb1.21200<br>bb121200 | 0.01.21200<br>bb121200 | '0-9','+-' | 121200 | If sign is defined, one + or - is allowed. Field here is defined with 8 characters. |
| C | 12.12.00 | 0.12.12.00 | '0-9','+-' | 121200 | If sign is defined, one + or - is allowed. |
| F, | 99bbbbb<br>bbbbb99 | 0990000<br>b990000 | '0-9,'+-' | 099 | If sign is defined, one + or - is allowed. |
| | | *** STRING FIELDS *** | | | |
| K | 099bbbb<br>099bbbb | 0990000<br>0990000 | '0-9','+-.'<br>'space' | 099 | Any combination of the legal char allowed. |
| L | AAbbb | not all. | 'A-Z','a-z'<br>'space' | AA | |
| M | 09Abbbb<br>09Abbbb | not all. | all printable<br>characters | 09A | |
| N | 0/99b/bbb<br>099bbbb | 0/990/000<br>0990000 | '0-9','+-.'<br>'space' | 099 | Any comb. of legal char is allowed. |
| O | b/AAA/bbb<br>bAAAbbb | not all. | 'A-Z','a-z',<br>'space' | bAAA | |
| P | A/-+-/Cbb<br>A-+-Cbb | not all. | all printable<br>characters | A-+-C | |
| Q | bb-+22,7<br>bb-+22,7 | 00-+22,7<br>00-+22,7 | '0-9','+-,.'<br>'space' | -+22,7 | Any comb of legal characters allowed. |
| R | bbbABCD<br>bbbABCD | not all. | 'A-Z','a-z'<br>'space' | ABCD | |
| S | b09-AXz<br>0b09-AXz | not all. | all printable<br>characters | | |

# APPENDIX B

# CONTROL CHARACTER HANDLING

NSHS is designed for asynchronus ASCII oriented video terminals with cursor control. Differences between terminal types occur at the control character level, i.e. characters with octal values between 1 and 37 inclusive.

In NSHS a "standard" control character set similar to the TDV 2000 control character set is used. For each terminal type, control characters must be translated to the NSHS set on input, and from the NSHS set on output.

The internal control character set is defined below: (I) indicates that the character value going into NSHS defines the function; (O) indicates the character value sent out by NSHS initiates the function.

| | | |
|---|---|---|
| 1 | Control A | Delete previous character (I) |
| 2 | Control B (X) | Defines blink display mode (NSD). Copies identical field above up to and including X (I) (NSL) |
| 3 | Control C | Copy one character from old line/field (I) |
| 4 | Control D | Copy all characters from old line/field without terminating (I) |
| 5 | Control E | Define field (NSD), insert characters in field (NSL) (I) |
| 6 | Control F | Abort picture creation/editing (I) (NSD). |
| 7 | Control G | Remove one line (NSD) (I), bell, i.e., audio signal (O) |
| 10 | Control H | Cursor left (I/O), ← or delete previous character if in a field or line (I) |
| 11 | Control I | Cursor right, → , (I/O) or copy old character if in field or line (I) |
| 12 | Control J | Insert one line (NSD) (I), skip to first field on next line in NSL and zero or blank out field if immediately after Control Q. (NSL) |
| 13 | Control K | Cursor down, ↓ , (I/O), or copy all characters old line or field and terminate (I) |
| 14 | Control L | Defines low intensity display mode (I) |
| 15 | Control M (CR) | Field and line terminator (I) |
| 16 | Control N (X) | Defines normal display mode (NSD) (I). Copies identical field above up to but not including X (NSL) (I) |

| | | |
|---|---|---|
| 17 | Control O (X) | Defines invisible ("off") display mode. Copies old field up to but not including X (NSL) (I) |
| 20 | Control P | Copies one character or field from previous line (NSD), or one character from identical field above (NSL) (I). Cursor addressing character (O), sets terminal in cursor address mode |
| 21 | Control Q | Deletes line or field (I), if given 3 times recreates picture. |
| 22. | Control R | Copies previous line (NSD) or identical field above (NSL) up to and including the last character without terminating (I). Represents normal display mode internally in NSHS (O) |
| 23 | Control S | Represents blink display mode internally in NSHS (O) |
| 24 | Control T | Represents low intensity display mode internally in NSHS (O) |
| 25 | Control U | Defines underline display mode (NSD). Represents underline display mode internally in NSHS (O) |
| 26 | Control V | Defines inverse video display mode (NSD) (I). Represents inverse video display mode internally in NSHS (O) |
| 27 | Control W | Represents invisible display mode internally in NSHS (O), used to terminate picture editing (I) in NSD. |
| 30 | Control X | Not used |
| 31 | Control Y | Erase screen (O) |
| 32 | Control Z (X) | Copy old field up to and including X (I) |
| 33 | Control Æ | Erase line [for output use only! This is ESCAPE!] (O) |
| 34 | Control Ø | Cursor up, ↑, (I/O) or copy all characters above line (NSD), or identical field (NSL) and terminate (I) |
| 36 | Control Å | Home, ↖ (I/O), and when in a line, copy previous field (NSD) (I) |
| 36 | | Alternative to Control W for terminating picture editing (I). |
| 37 | | Clear whole line (0), (NSL). |

NSHS can accommodate up to 8 different types of terminals simultaneously. New VDU types can also be implemented. If you intend to use new VDU types you should contact your local ND support organisation for advice in this matter.

# APPENDIX C

# ERROR MESSAGES AND CODES

The following error messages can occur in the NORD Screen Handling system:

- "no such picture"
- "no pictures defined on file"
- "file not successfully opened"
- "error in reading first block of file"
- "error in writing first block of file"
- "error in closing scratched file"
- "first block of file damaged"
- "block size on file in error"
- "error in writing block back to file"
- "array not big enough for picture"
- "picture name table on file full"
- "entry number greater than number of pictures (internal error!)"
- "error in reading block from file"
- "first block of picture is not first block in chain"
- "not enough blocks available on file"

These error messages are written out with some tracing information;

<["possible file system error text"]
"error position identifiers are: 99999. 99999 source/object file"
"error message">

The tracing information can be particularly helpful in locating a bug in NSD.

In the NORD Screen Library system, error messages are printed on the last line of the terminal and only during the execution of an RFLDS call. These messages are in English and can be changed through editing by your local ND support organisation.

For all subroutine calls the return status parameter "IST" can have the values:

0: call successfully executed

1: unknown picture name or number

—N: field number/indicator or picture number/name data-element-index number N is in error

2: other errors

For other errors and at times for 1 or —N, an additional error code is placed in ITERM(7). These error codes are as follows:

| *Value:* | *Meaning* |
|---|---|
| 1 | terminal common array in error |
| 2 | illegal picture number |
| 3 | no picture with this number in buffer |
| 4 | first field indicator is zero, but number of field indicators is not 1 |
| 5 | number of field indicators < 0 or > 255 |
| 6 | line number in a field indicator is too large |
| 7 | no fields defined on this line |
| 8 | no fields defined on this picture |
| 9 | not so many fields on line |
| 10 | field number is zero |
| 11 | absolute field number > number of fields in picture |
| 12 | field numbers in wrong order |
| 13 | picture buffer full?? |
| 14 | too many pictures requested, or buffer full |
| 15 | number of pictures in error |
| 16 | picture buffer damaged |
| 17 | too many field numbers given |
| 18+100*EN | error in reading or opening picture file, here EN is the NORD file system error number. For example 66 18 is file system error 66 (decimal), 102 octal |
| 19 | no pictures defined on this picture file |
| 20 | not enough space in buffer for this picture |
| 21 | data element index array in error |
| 22 | picture file block damaged |

| | |
|---|---|
| 23 | picture description must not start at address zero |
| 24 | parameter "code" in error |
| 25 | number of field numbers < or=0 |
| 26 | all fields locked on an RFLDS or WFLDS call |
| 27 | line number in error |
| 28 | position number in error |
| 29 | area mode does not allow reading/removal of pictures (IPUBL(1) or IPRIV(1) ´ 0) |
| 30 | no pictures in private buffer |
| 31 | incorrect program mode |
| 32 | no such picture file |
| 33 | user control illegal |
| 35 | Error in input monitor call, Sintran error in ITERM(127) |
| 50 | area mode in error |

# APPENDIX D

# SYSTEM CONTROL

System control is called from pictures when field control no = 8.

*There are 6 system control functions implemented:*

1 Person Number
2 Post Office Account Number - modus 10
3 Post Office/Bank Account Number - modus 11
4 Date (Year, month, day)
5 Date (Day, month, year)
6 Date (Month, day, year)

1. Person Number

Person Number must consist of 11 digits following the standard:

99 99 99 999 99

where the digit groups mean:

day of birth
month of birth
year of birth
individual digits (last digit indicates sex)
control digits

2. Post Office Account - Modus 10

The Post Office Account Number consists of up to 20 digits where the last digit is the control digit.

3. Post Office/Bank Account - Modus 11

The Post Office Account Number consists of up to 20 digits where the last digit is the control digit. The Bank Account Number must consist of 11 digits where the last digit is the control digit.

*Note that the vector digits of points 1, 2 and 3 follow the Norwegian standard.*

The default weight digits for modus 11 are 2,3,4,5,6,7,2,3,4 ...counted from right to left. This means that the last digit should be multiplied by 2; the one before by 3, etc. You may replace the existing digits in modus 11 by including your own weight digits. To define your own weight digits, a routine of one common block must be declared and loaded *after* the NSL.

The layout of the routine must be as follows:

```
      BLOCK DATA
      COMMON/PICDEF/IUSVEC (10)
      DATA IUSVEC/1003B, 2005B, 3007B, 1003B, 2005B, 3007B,
—     1003B, 2005B, 3007B, 11003B/
      END
```

Note that the weight digits are stored in octal with two digits in each word, and are therefore best declared octally.

*Example*:

If you wish to have the following weight digits 1,2,3,4,5,6,7,1,2,3... in your system check on modus 11,
the "PICDEF" common will be:

```
      BLOCK DATA
      COMMON/PICDEF/IUSVEC (10)
      DATA IUSVEC/402B, 1404B, 2406B, 3401B, 1003B, 2005B, 3007B,
—     402B, 1404B, 2406B/
      END
```

### 4,5,6 Date

If the year is within the range 1900 - 1999, the last two digits are sufficient, otherwise, all four digits (i.e., 1876) must be used.

Six additional date control numbers are used for the date system control to check the legality of the date input:

1 = date must equal today (i.e., equal to what SINTRAN III believes the date to be)

2 = date must be equal to or after today

3 = date must be after today

4 = date must be before or equal to today

5 = date must be before today

6 = no check of date

# APPENDIX E

# TROUBLE SHOOTING

1. If the pictures came out incorrectly on the VDU screen:

   a) Have you set the correct terminal type and the required picture displacement? Check ITERM(2).

   b) If TDV 2000, is it in page or roll mode and if so, have you set bit 15 of the terminal type?

   c) Have you set break strategy to 0? If not, do you have a SINTRAN III system with a version date later than 1977?

   d) If INFOTON 200, is the terminal in Page Mode?

*Otherwise*

2. Have you checked the status value returned by all the subroutine calls? If this status is not Ø, have you checked the value of the error code in ITERM(7)?

   A common error code is 6618, which means that the user calling GTPIC does not have a read and common access to the picture file.

   Another common error is 6616: Picture buffer damaged. Probably too little space has been allocated for one of the buffer (integer) arrays.

3. Have you set up the terminal buffer, the private buffer and the public buffer correctly?

4. If your program has inexplicable or mysterious errors, are you explicitly using any of the "internal" routines in the NSL? If so, are these "assembly" routines and have you declared them as such in all subroutines which use them?

5. Both for Fortran and Cobol programs, there are very efficient tools for debugging. When the compiler is called, the first command should be 'DEBUG'.

For example for a Fortran program:

```
@FTN
$DEBUG
$COMP.....
.
.
.
```

and similarly for a Cobol program. The debugging procedures are described in the manuals.

# APPENDIX F

# USER CONTROL AND LOW-LEVEL ROUTINES, AN EXAMPLE.

*Description of the UCONT routine:*

```
SUBROUTINE UCONT(IUSCOD,IPN,IFINFO,NOF,FNA,REC,DEIA,
INDEX,RBITS,IST)


C      IUSCOD  : THE USER CONTROL NUMBER.
C      IPN     : THE PICTURE NUMBER          (Same as in the RFLDS call).
C      IFINFO  : FIELD INFORMATION ARRAY.
C      NOF     : NUMBER OF FIELDS            (Same as in the RFLDS call).
C      FNA     : FIELD NUMBER ARRAY          (Same as in the RFLDS call).
C      REC     : RECORD                      (Same as in the RFLDS call).
C      DEIA    : DATA ELEMENT INDEX ARRAY    (Same as in the RFLDS call).
C      INDEX   : INDEX TO THIS FIELD NUMBER.
C      RBITS   : BIT ARRAY WITH BIT SET IF FIELD READ.
C      IST     : INPUT/RETURN STATUS.

          INTEGER FNA(1),REC(1),DEIA(1),RBITS(1),IFINFO(1)
                          = = = || = = =
```

When the system encounters a field that has been assigned Field Control Function 9 (user defined control algorithm), the subroutine UCONT is called automatically by RFLDS both before the field is entered and after a value has been given. When UCONT is activated, its status parameter IST on input indicates if we enter or leave the field.

IST = –1: The field is going to be read after return from UCONT.
IST = 1: The field is read, and the field value is stored in the RECORD.

On return IST should have the following values:

IST = –1: Read the same field.
IST = 0: Write the field contents found in the record to the screen, continue to the next field to be read.
IST = 1: Just continue to the next field to be read. The field contents is not written to the screen.
IST > 99: RFLDS terminates with terminating character = -1 and returns status equal IST. No further action is taken with the field contents.
IST < –1: Same as if -1 is given
IST = <1,99>: Same as if 1 is given.

On return from UCONT, RFLDS checks the error code in ITERM(7), and if this is not 0, RFLDS will terminate.

With the parameter INDEX, the UCONT routine can decide which field is the next to be read by RFLDS. On input the parameter contains the index to the current field. On return, index to next field to be read can be given. If the index is outside the range of fields (not in the area <1,number of fields>), the first field in the picture is read. This works only if the status IST on return is 0 or 1.

*How to load the UCONT routine:*

In the Screen Library there is a default UCONT routine which only returns a proper status value. Any user-written UCONT must therefore be loaded before the Screen-Library, and the user must make sure that his routine really is loaded. On using the NORD-RELOCATING- LOADER he can use the command X-LOAD. The loading sequence for a Fortran program will then be:

```
@NRL
LOAD APPLICATION
X-LOAD UCONT
LOAD SCREEN-LIBRARY
LOAD FTNLIB
DUMP PROGRAM
EXIT
```

AN EXAMPLE OF AN UCONT ROUTINE:

Program listing:

```
      NORD-10/ND-100 FORTRAN COMPILER FTN-2090I
      SUBROUTINE UCONT(IUSCOD,IPN,IFINFO,NOF,FNA,REC
                              ,DEIA,INDEX,RBITS,IST)
C        THE ROUTINE IS USED FOR AUTOMATIC TRANSFER OF FIELD VALUES
C        FROM ONE RECORD TO THE NEXT IN A LOOP-SEQUENCE
C        WHERE INPUT-RECORDS ARE READ FROM THE SAME PICTURE.
C
C     WHERE:

C     IUSCOD   : THE USER CONTROL NUMBER
C     IPN      : THE PICTURE NUMBER
C     IFINFO   : FIELD INFORMATION ARRAY
C     NOF      : NUMBER OF FIELDS
C     FNA      : FIELD NUMBER ARRAY
C     REC      : RECORD
C     DEIA     : DATA ELEMENT INDEX ARRAY
C     INDEX    : INDEX TO THIS FIELD NUMBER
C     RBITS    : BIT ARRAY WITH BIT SET IF FIELD READ
C     IST      : RETURN STATUS


      INTEGER FNA(1),REC(1),DEIA(1),RBITS(1),IFINFO(1)
      INTEGER IFINFO(22),PINCH
      ASSEMBLY GFINF,MCURS,PINCH,POUTC,IPNUM

C                   *** INPUT-STATUS IS -1 BEFORE ANY VALUE IS
C                   *** GIVEN TO THE FIELD. NO ACTION IS TAKEN.
      IF (IST .EQ. -1) GOTO 90

      IADDR = IPNUM(IPN)
   10 IACT  = DEIA(INDEX)
      IDATA = REC(IACT)

      IF (IUSCOD .NE. 1) GOTO 30
C====================================
C     OR MORE GENERAL:              =
C     GOTO (1,2,3,4,......,255) IUSCOD     =
C     CONTINUE                      =
C        ** CONTROL 1; GO TO RETURN =
C     CONTINUE                      =
C        ** CONTROL 2; GOTO RETURN  =
C                                   =
C                                   =
C                                   =
C55   CONTINUE                      =
C        ** CONTROL 255; GOTO RETURN =
C====================================

C                   *** REST OF THE CODE IS ACTIVATED ONLY IF
C                   *** USER-CONTROL NUMBER IS 1.
      IDATA = ISHFT(IDATA,-8).AND.177B
C                   *** IDATA = INPUT VALUE IN FIELD.
C                   *** 111B = "I",114B = "L"
      IF (IDATA .EQ. 111B)     GOTO 70
      IF (IDATA .EQ. 114B)    GOTO 30
C                 *** FOR ALL OTHER VALUES, JUST RETURN.
      GOTO 80

30    CALL WMSGE('TYPE L IF LOCK REQUIRED,IF NOT,USE:CURSOR RIGHT,IF FIN
     -ISHED, USE CURSOR HOME*')

      J=0
C                ** LOOP FOR TREATING EACH FIELD. THE LOOP INDEX J IS
C                ** CALCULATED FROM INPUT VALUES.
40    J = J + 1
      IF (J .LE. 0) J = 1
      IF (J .GT. N) GOTO 70
C                 *** HERE J IS BETWEN 1 AND NUMBER OF FIELDS.
C                 *** GET INFORMATION ABOUT FIELD, PLACE CURSOR
C                 *** AT BEGINNING OF FIELD, READ ONE CHARACTER.
45    IFN = FNA(J)
      CALL GFINF(IFINFO,IFN,IADDR)
      CALL MCURS(IFINFO(11),IFINFO(12))
50    ICHAR = PINCH(1)
C                 *** CHARACTER READ TO ICHAR.
      IF (ICHAR .EQ. 11B) THEN
C                 *** CURSOR RIGHT: RESET EVENTUALLY
C                 *** LOCK ON FIELD, WRITE FIELD WITH
C                 *** NORMAL DISPLAY, GO TO NEXT FIELD.
         CALL RLOCK(IPN,1,J,FNA,IST)
         CALL WFLDS(0,IPN,1,FNA(J),REC,DEIA(J),IST)
         GOTO 40
      ELSEIF (ICHAR .EQ. 114B) THEN
C                 *** L: WRITE FIELD WITH LOW INTENSITY,
C                 *** LOCK FIELD, GO TO NEXT FIELD.

         CALL WFLDS(3,IPN,1,FNA(J),REC,DEIA(J),IST)
         CALL ZLOCK(IPN,1,J,FNA,IST)
         GOTO 40

      ELSEIF (ICHAR .EQ. 34B) THEN
C                 *** CURSOR UP FOR FINISH.
         GOTO 70
      ELSEIF (ICHAR .EQ. 10B) THEN
C                 *** CURSOR LEFT FOR STEP BACK TO
C                 *** PREVIOUS FIELD.
         J = J - 2
         GOTO 40
      ENDIF
C               *** HERE IF ILLEGAL INPUT GIVEN, BELL ON TERMINAL,
C               *** GO TO NEW INPUT.
      CALL ZBELL
      GOTO 50

70    CONTINUE
C     ** FINISH SEQUENCE: WRITE MESSAGE TO LAST LINE OF TERMINAL.
C     ** PLACE CURSOR ON FUNCTION FIELD, READ ONE CHARACTER.

      CALL WMSGE('FUNCTION FIELD TO BE LOCKED ? (Y OR N)*')
      CALL MCURS(IFINFO(11),IFINFO(12))
      ICHAR = PINCH(1)
```

```
C             *** CHARACTER READ TO ICHAR.
    IF (ICHAR .EQ. 131B) THEN
C             *** Y FOR YES, ECHO CHARACTER,
C             *** LOCK FUNCTION FIELD.

        CALL POUTC(1,131B)
        CALL ZLOCK(IPN,1,INDEX,FNA,IST)
    ELSEIF (ICHAR .EQ. 10B) THEN
C             *** CURSOR LEFT: GO TO PREVIOUS FIELD.
        J = J - 2
        GOTO 45
    ELSEIF (ICHAR .EQ. 116B) THEN
C             *** N FOR NO: ECHO CHARACTER, RETURN
        CALL POUTC(1,116B)
    ELSE
C             *** ILLEGAL INPUT, TRY AGAIN.
        GOTO 70
    ENDIF

80  IST = 0
    RETURN

90  IST = -1
    RETURN
    END
```

## *Description of low-level routines which can be called from an application program or a UCONT-routine*

Available routines:

ICHEK : get index to picture description, check field-number-array, check/generate data-element-index-array.
GFINF : get all system information about a field.
MCURS : place the curser on given position on the screen.
PINCH : read a character from the terminal.
POUTC : write a character to the terminal.
SBYTE : store a byte to an array.
LBYTE : load a byte from an array.
ITBIT : test if a bit in an array is set or not.
IZBIT : zero a bit in an array.
ISBIT : set a bit in an array to one.

For the routines ICHEK, GFINF, MCURS, PINCH and POUTC the common-area PRIVATE must be defined and correctly initialized as defined earlier in the manual.

Description of the routine:

ICHEK:

Call sequence:

```
IST = ICHEK(code,picture-number,field-number-array,data-
                        element-index-array,picture-core-index)
```

Parameters:

code = 0; returns core address to picture description in picture-core-index, checks field-number-array. If any errors,error codes are found in IST and ITERM(7).

code >< 0; same as if code = 0 and in addition:
data-element-index-array(1) = 0; generates the values in data-element-index-array.
data-element-index-array(1) ><0; check if the values in data-element-index-array is in the range of 1 to 2048.

IF no errors found, IST = 0 on return.

GFINF:

Call sequence:

```
ASSEMBLY GFINF
INTEGER ARRAY  field-information-array(23)

IST = ICHEK(code,picture-number,field-number-array,data-
           element-index-array,picture-core-index)
IF ( IST .NE. 0 ) GOTO ERROR

CALL GFINF(field-information-array,field-number, picture-core-index)
```

MCURS:

call sequence:
ASSEMBLY MCURS

CALL MCURS (line-number,position-on-line)
line-number: from 1 to number of lines on this terminal type.
position-on-line: from 1 to no. of positions on line.

If the user tries to position the cursor outside the defind area, the cursor is placed in home-position.

PINCH:

Call sequence:
INTEGER PINCH
ASSEMBLY PINCH

input-character = PINCH(device-number)
Not printable characters are translated to values as given in appendix B.

POUTC:

Call sequence:
ASSEMBLY POUTC

CALL POUTC(device-number,character-value)
Only printable characters must be written to the terminal with this routine. The characters are buffered in groups of 8 before they are sent to the output device with monitor call MON 22.
If the user wants an uncompleted buffer to be transmitted, he must call POUTC with character-value = zero. Output strategy may be changed to use MON 2 and no buffering by using the status/option facillity, see appendix-G.

SBYTE:

Call sequence:
ASSEMBLY SBYTE

CALL SBYTE(array,byte-number,byte-value)
The bytes in "array" is numbered from 1 to N. The rightmost byte in "byte-value" is stored.

LBYTE:

Call sequence:
ASSEMBLY LBYTE

byte-value = LBYTE(array,byte-number)
The bytes in "array" ist numbered from 1 to N.

ITBIT IZBIT ISBIT are used for bit operations on an integer array. The bits in the array are numbered:

ARRAY(1): 15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
ARRAY(2): 31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16
ARRAY(3): 47,46,45,....................................
etc.

Call sequence:
ASSEMBLY ITBIT, IZBIT, ISBIT

CALL IZBIT(array,bit-number)
CALL ISBIT(array,bit-number)
bit-value = ITBIT(array,bit-number)

Bit-value gets the value 0 or 1.

*Description of data returned in field-information-array:*

(01); Number of characters read to field.
(02); Fill code 40B (space) or 60B (zero).
(03); Sign code: —1 => no sign defined.
55B => field is negative.
0 => field has positive value.
40B => field has positive value.
(04); Edit code: 1 - 9, edit code 1-9
10, edit code 0
11-29, edit code A-S
(05); Number of significant characters. Not including edit characters.
(06); Number of positions that field occupies in picture.
(07); Storage code according to defined storage 1 - 5.
(08); Logical device number for terminal.
(09); Code for termination of input to field:

| | |
|---|---|
| —4 | CTR(Q) CTR(A) |
| —2 | CTR(Q) CTR(Q) CTR(Q) |
| —1 | ESCAPE AS DEFINED IN ITERM(3) |
| 0 | CR, CTR(Q) CTR(J) (puts empty value to field), CTR(R) (copies value of identical field above) |
| 1 | USER DEFINED FIELD TERMINATOR |
| 2 | CURSOR LEFT |
| 3 | CURSOR RIGHT |
| 4 | CURSOR DOWN |
| 5 | CURSOR UP |
| 6 | HOME |
| 7 | CTR(L) |
| 8 | CTR(S) |
| 9 | CTR(T) |
| 10 | CTR(U) |
| 11 | CTR(V) |
| 12 | CTR(W) |
| 13 | CTR(X) |
| 14 | CTR(Y) |
| 27 | LF |

(10); > <0 if field is must-read field and is not read before.
(11); line number of field.
(12); column number of field.
(13); field-number of identical field above if any.
(14); 1 if control read, else 0.
(15); no display flag, if 1 then display modes are dropped.
(16); lenght of data element in words.
(17); control type, 0 if none.
(18); number of words of control information.
(19); address of control information.
(20); code for internal escape character as defined in ITERM(3)
(21); read strategy as in ITERM(4)
(22); break strategy as in ITERM(8)
(23); bit 0 set if CR must be given twice for zero field.

# APPENDIX - G

# THE STATUS/OPTION WORD ITERM(9) AND SYSTEM DESCRIPTION OF ITERM(11) - ITERM(128)

## STATUS/OPTION WORD ITERM(9):

Experience shows that the contents of this word is changed with new releases of NSHS. It is therefore important to study the Program Description sheets and the Revision log for the latest release.

The meaning of each bit in the word is:

BIT 0 If bit is set, it is possible to force illegal values to fields defined with control. When an illegal value is given, an error message is written on the message line and the cursor is set to the first position of the field. Giving "CURSOR-RIGHT" then forces the illegal value into the field.

BIT 1 This bit is for internal use in RFLDS.

BIT 2 If the bit is set before a call to RFLDS, RFLDS clears the message line and sets the bit to zero before any fields are read. If bit is zero on input to RFLDS, no action is taken.

BIT 3 If the bit is set, output-strategy as defined in the next three bits,4, 5, and 6, is chosen. The next three bits are tested in succession and output strategy is determined by the first bit different from zero. If bit 3 is zero, the monitor call MON 22 is used and NSHS buffers 8 characters before they are transmitted.

BIT 4 If bit 3 and this bit is set to 1, MON 2 (outbyte) is used.

BIT 5 If bit 3 and this bit is set to 1 and bit 4 is zero, all output from NSHS is written to a buffer in memory. The address of the buffer must be defined in ITERM(30) and the size of the buffer (in bytes) in ITERM(33). ITERM(32) holds the number of bytes written to the buffer and must be set to zero before any writing starts. When the buffer is full,an assembly routine QERR is called. On input to this routine the A-register = —1 and the T-register contains a return address to NSHS. The routine is explained later in this appendix.

BIT 6 If bit 3 = bit 6 = 1 and bit 4 = bit 5 = 0, MON 162 (outstring) is used. The output is buffered with 182 characters before transmission.

BIT 7 If bit is set, "home" given in first position of a field, will terminate RFLDS. The terminating code on return will be 100. Status and ITERM(7) is set to zero.

BIT 8 If bit is set, the UCONT-routine will automatically be called for all fields. The User-control-number is set to 1000. Defined User-control-numbers are supressed.

BIT 9 If bit is set, the user can decide which lines of a picture are to be written by WRPTD/WRPTF. Iterm(126) must be set to the first line, Iterm(127) to the number of following lines to be written. Iterm(126) and Iterm(127) are set to zero after each Write. If unreasonable values are given, the whole picture is written.

BIT 10 If bit is set, error in input monitor call will terminate RFLDS, but not abort the program. Error code Iterm(7) is set to 35, and the error code from the monitor call is found in ITERM(127).

BIT 11 If the bit is set, it is possible to terminate input to a field with CTRL/L-Y which are then used as function keys. The terminating code is transferred to UCONT in IFINFO (9) and UCONT takes appropriate action, resets the control value and returns.

BIT 12 If bit is set to one, fields with "Must-read" and default value defined can be passed by giving only CR. The default value is put to the field and the cursor continues to the next field to be read.

BIT 13 Bit set gives special processing of fields with old contents and the read-bit set. If the operator gives CR in the first position of these fields, RFLDS will answer with bell to the terminal. If the field really should have zero value, the operator must type CR a second time. If bit 15 is zero, CR gives the field zero value the first time it is typed.

BIT 14 If the bit is one, the effect of CR and SPACE are interchanged as terminating character for decimal fields (edit code 0 - 9).

BIT 15 Bit set, WRPTD writes spaces in field positions instead of dots.The status option continues in ITERM(125):

> If bit 0 is set, WRPTD and WRPTF writes arrow rights instead of spaces. Bits 1-15 are not used.

System description of the ITERM array:

```
%          ************** I T E R M ( 1 2 8 ) ********************
%          *                                                        *
%          * I--- DESCRIBED IN NSHS MANUAL --------------------I    *
%          * I                                                 I    *
% 000   1  * I   DEVICE NUMBER                                 I    *
% 001   2  * I   PICTURE DISPLACEMENT/TERMINAL TYPE            I    *
% 002   3  * I   ESCAPE                                        I    *
% 003   4  * I   RETURN STRATEGY                               I    *
% 004   5  * I   PROGRAM MODE                                  I    *
% 005   6  * I   CURSOR POSITION                               I    *
% 006   7  * I   ERROR CODE                                    I    *
% 007   8  * I   BREAK STRATEGY                                I    *
% 010   9  * I   STATUS/OPTION FLAG                            I    *
% 011  10  * I   PRIVATE LENGTH                                I    *
%      --  * ---------------------------------------------------    *
% 012  11  *     SL1REG   PINCH/POUTC - LREG                        *
% 013  12  *     SL2REG   DMCURS - LREG                             *
% 014  13  *     SL3REG   INBYT/UTBYT - LREG                        *
% 015  14  *     SCHREG   POUTC  - CHARACTER                        *
% 016  15  *     SDVREG   POUTC  - DEV.NO.                          *
% 017  16  *     SAREG    INBYT/UTBYT SAVE A-REG                    *
% 020  17  *     STREG    INBYT/UTBYT SAVE T-REG                    *
% 021  18  *     SDREG    INBYT/UTBYT SAVE D-REG                    *
% 022  19  *     SXREG    INBYT/UTBYT SAVE X-REG                    *
% 023  20  *     SWCREG   UTBYT COUNTER                             *
% 024  21  *     S8A      8 BYTES OUTBT A INFORMATION               *
% 025  22  *     S8D      8 BYTES OUTBT D INFORMATION               *
% 026  23  *     S8L      8 BYTES OUTBT L INFORMATION               *
% 027  24  *     S8X      8 BYTES OUTBT X INFORMATION               *
% 030  25  *     S81R     WRPTF  SAVED  L INFORMATION               *
% 031  26  *     S82R     SAVED CURSOR ADDRESS FOR DMCURS           *
% 032  27  *     S83R     SAVE LREG WHEN CALLING DMCURS FROM POUTC*
% 033  28  *     S84R     POINTER TO PAR. LIST FOR OUTST.           *
% 034  29  *     S85R     POINTER TO DEVICE NUMBER                  *
% 035  30  *     S86R     POINTER TO MEMORY ADDRESS OF BUFFER       *
% 036  31  *     S87R     POINTER TO NO. OF CHARACTERS              *
% 037  32  *     S88R     NO. OF CHARACTERS                         *
% 040  33  *     S89R     USED OF RMOUT/SMOUT                       *
% 041  34  *     BUFFER   START                                     *
%   .      *  .                                                     *
%   .      *  .                                                     *
% 172 123  *     BUFFER END                                         *
% 173 124  *              INDEX TO CURRENT FIELD IN RFLDS           *
% 174 125  *     STATT    CONTINUATION OF ITERM(9):                 *
%          *              BIT 0 : 1= WRPTD/WRPTF WRITES PACKED      *
%          *                         SPACES AS COURSORS RIGHT       *
%          *                    0 : 0= SPACES ARE USED              *
% 175 126  *     USED OF WRPTD/F FOR START NR. OF LINES TO BE       *
%          *     WRITTEN                                            *
% 176 127  *     NR. OF LINES TO BE WRITTEN.                        *
%          *                    OR                                  *
%          *     CONTAINS THE ERRORCODE FROM PINBT (MON 1).         *
%          *     IS CLEARD BEFOR CALL TO PINBT.                     *
%          *                                                        *
% 177 128  *     STELL   OUTPUT CHARACTER COUNTER                   *
%          *                                                        *
%          **********************************************************
%
%      STATUS/OPTION FLAG:
%
```

```
%    ITERM(9):
%
%       BIT  0 = IGNORE CONTROL
%       BIT  1 = INTERNAL STANSAAB FLAG
%       BIT  2 = INTERNAL MESSAGE FLAG
%       BIT  3 = SPECIAL OUTPUT STRATEGY  (MCN 22 IS DEFAULT)
%       BIT  4 = USE MON 2; OUTBYT
%       BIT  5 = WRITE TO MEMORY BUFFER
%       BIT  6 = USE MON 162; OUTSTRING
%
%       BIT  7 = "HOME" TERMINATES RFLDS WITH TERM-CHAR = 100.
%       BIT 10 = 'UCONT' IS ALWAYS CALLED FOR ALL FIELDS.'UCNR' = 1000
%       BIT 11 = USED WHEN WRPTD/F IS TO WRITE ONLY PORTIONS OF PICTURE
%                ITERM(126) = START LINE
%                ITERM(127) = NO. OF LINES
%                BIT 11 , ITERM(126) AND ITERM(127) ARE RESET AFTER CALL
%       BIT 12 = SET IF ERROR FROM MON 1 SHALL TERMINATE RFLDS.
%       BIT 13 = BIT SET, CTR/L-Y TERMINATES INPUT TO FIELD. TERMINATING
%                CODE IS TRANSFERED TO UCONT IN IFINFO(9).
%       BIT 14 = BIT SET, CR IN FIRST POSITION OF FIELD WITH MUST-READ
%                         GIVES FIELD DEFAULT VALUE AND RFLDS CONTINUES
%                         TO NEXT FIELD.
%       BIT 15 = BIT SET, MAKES IT DIFFICULT TO ZERO FIELD WITH CR.
%       BIT 16 = DESIMAL-FIELD FLAG. 0; CR RIGHT JUSTIFY  FIELD
%                                       SP JUSTIFY TOVARDS COMMA
%                                    1; OPOSITE.
%       BIT 17 = BIT SET, SPACE IS WRITTEN IN FIELDS IN STEAD OF DOTS
%                         WHEN WRPTD IS USED.
%
%    ITERM(125):
%
%       BIT 00 = BIT SET, WRPTD/WRPTF USE CORSOR RIGHTS IN STEAD OF
%                SPACES
%       BIT 01 - 17  NOT USED
%
%.....................................................................
```

*Description of the routine QERR:*

Global symbols defined in Screen-libraries:

```
    DISP 176;
    INTEGER INERR;
    PSID;
    INTEGER NUNIT:=2,TUNIT:=2;
    INTEGER ARRAY M104:=(NUNIT,TUNIT);
```

The routine definition:

```
    *)9BEG; )9ENT QERR )9LIB QERR )9EXT QUIT

    SUBR QERR

    INTEGER ARRAY TTNUM(0); *TNN=0; )9FABS TNN; *)9ADS PRIVATE TNN
    INTEGER POINTER ITERM=TTNUM

    QERR:  L=:D:T=:L     % L and T are return-addresses. Return to T if
                         % the output is to be repeted, to L if the
                         % execution is to continue. A contains the error
                         % code from a monitor call or -1 if memory output
                         % buffer is full.
           IF A = 161    THEN
                         % A=161 means no answer from remote
                         % computer, wait two seconds and
                         % try again.
               A:="M104"; *MON 104
               EXIT
           ELSE
               IF T:="ITERM". STATO BIT 12 THEN      % Put error code in
                                                     % ITERM(127) and
                                                     % let RFLDS
                                                     % terminate.

                 A=:"ITERM".INERR
                 L:=D
                 EXIT
               ELSE                % write sintran error to terminal,
                                   % go back to sintran.
                 * MON 64
                 CALL  QUIT
               FI
           FI
    RBUS
    *)9END

    *)9BEG
    *)9ENT  QUIT
    *)9LIB  QUIT
    SUBR  QUIT
    QUIT: *MON 0
          EXIT
    RBUS
    *)9END
```

# APPENDIX H

# NSL AND COBOL

## COMMON AREAS

If an application program is written in COBOL, a FORTRAN BLOCK DATA subroutine *must* be loaded before NSL and normally after the COBOL program.

The BLOCK DATA subroutine dimensions and initiates the NSL Private and Public common areas.

*Example:*

```
BLOCK DATA
COMMON/PRIVATE/ITERM(128),IPRIV(1920)
COMMON/PUBLIC/IPUBL(6)
DATA ITERM/1,3,100007B,0,0,0,0,0,0,1920,118*0/
DATA IPRIV/0,1920,3,0,0,1915*0/
END
```

The length and contents of IPRIV and the contents of the *first* 10 words of ITERM, are used here solely as an example. The contents of ITERM and IPRIV can be changed from a COBOL program by calling the subroutine COSCR1.

```
          SUBROUTINE COSCR1(IARR)

C    PARAMETERS:
C    INTEGER ARRAY IARR(8),
C    IARR(1) = DEVICE NR. FOR TERMINAL
C    IARR(2) = TERMINAL TYPE AND PICTURE DISPLACEMENT
C    IARR(3) = ESCAPE CHARACTER
C    IARR(4) = READ STRATEGY
C    IARR(5) = PROGRAM MODE
C    IARR(6) = BREAK AND ECHO STRATEGY
C    IARR(7) = OPTION WORD
C    IARR(8) = MAX NR OF PICTURES

          INTEGER IARR(1)
          COMMON/PRIVATE/ITERM(128),IPRIV(30)
          COMMON/PUBLIC/IPUBL(6)
          ITERM(1)=IARR(1)
          ITERM(2)=IARR(2)
          ITERM(3)=IARR(3)
          ITERM(4)=IARR(4)
          ITERM(5)=IARR(5)
          ITERM(8)=IARR(6)
          ITERM(9)=IARR(7)
          IPRIV(3)=IARR(8)
          RETURN
          END
```

*Note: Only relevant data can be changed. ITERM(6), ITERM(7), ITERM(10), IPRIV(1), IPRIV(2), IPRIV(4) and IPRIV(5) can not be changed with COSCR1.*

Values for ITERM(6) and ITERM(7) are returned from the subroutine ERRORC.

```
      SUBROUTINE ERRORC(ITERM7,ITERM6)
      COMMON/PRIVATE/ITERM(128)
C     ITERM7:  NSL ERROR CODE  ITERM(7)
C     ITERM6:  CURSOR POSITION  ITERM(6)

      ITERM7 = ITERM(7)
      ITERM6 = ITERM(6)
      RETURN
      END
```

## COBOL—FORTRAN INTERFACE

In COBOL there is no data type that corresponds to the FORTRAN data type "CHARACTER". Consequently, new entry points to some of the FORTRAN subroutines have been designed especially to overcome this language incompatability. These routines can be distinguished by their suffix, the letter C (ERRORC, OPENFC, CLOSEC, GTPICC, WMSGEC).

A file is opened in a COBOL program by using the normal OPEN INPUT/OUTPUT procedures. For special purposes we may also call the subroutine OPENFC:

```
                  SUBROUTINE   OPENFC(IPNAS,IACC,IFTNR,IST)
                  INTEGER   IPNAS(1),IFNAM(20)
                  CHARACTER    NAME*40
                  CHARACTER    ACS*2
                  EQUIVALENCE    (IFNAM(1),NAME)

     C            IPNAS:  FILE NAME STRING.
     C            IACC  :  ACCESS
     C                       0 = SEQUENTIAL WRITE
     C                       1 = SEQUENTIAL WRITE APPEND.
     C            IFTNR:  FORTRAN FILE NUMBER.
     C            IST   : RETURN STATUS
     C                       0 =OK.
     C                      —1 = ERROR IN PARAMETER IACC.
     C                       OTHER  = FORTRAN ERRCODE.


                  IF (IACC.NE.0.AND.IACC.NE.1) GOTO 91
                  ACS = 'W'
                  IF (IACC .EQ. 1) ACS = 'WA'

                  DO  10 I = 1,20
     10           IFNAM(I) = IPNAS(I)
                  OPEN (IFTNR,FILE=NAME,ACCESS=ACS,
                  STATUS = 'UNKNOWN',ERR=92)
                  IST = 0
                  GOTO  999
     91           IST = —1
                  GOTO  999
     92           IST = ERRCODE
     999          RETURN
                  END
```

The closing procedure corresponding to OPENFC, is CLOSEC:

```
              SUBROUTINE CLOSEC(IFTNR)

C             IFTNR:  FORTRAN FILE NUMBER.
C                     —1  =  CLOSE ALL OPENED FILES.

              CLOSE (IFTNR)
              RETURN
              END
```

The COBOL interface routines for WMSGE and GTPIC. Observe that the Cobol string variables for picture-names, picture-file-name and message-string must have exactly the same length as corresponding character-variables in the interface routines.

```
              SUBROUTINE WMSGEC(ISTR)
              INTEGER  ISTR(1),IKAR(41)
              CHARACTER  ICHAR*80
              EQUIVALENCE  (IKAR(1),ICHAR)
              DO  10  I=1,41
10            IKAR(I)=ISIR(I)
              CALL WMSGE(ICHAR)
              RETURN
              END


              SUBROUTINE  GTPICC(IPFN,NOPC,INAM,IPNA,IST)
              INTEGER  IPFN(1),INAM(1),IPNA(1)
              DIMENSION  MFINA(21),MPINA(65)
              CHARACTER  IFIL*40,NAME*128
              EQUIVALENCE  (MFINA(1),IFIL),(MPINA(1),NAME)
              DO  10  J=1,21
10            MFINA(J)=IPFN(J)
              DO  20  J=1,65
20            MPINA(J)=INAM(J)
              CALL  GTPIC(IFIL,NOPC,NAME,IPNA,IST)
              RETURN
              END
```

*Unused Byte Positions*

Cobol programmers should note that unlike COBOL where a field is byte-oriented, NSL is *word-oriented.* In other words, in NSL, fields can contain unused bytes. This situation will occur for example with Storage Code 3, if the number of significant characters in addition to a possible sign renders an uneven number of bytes. When programming in COBOL therefore, it is wise to define byte fields such that they use an *even* number of positions. The unused position is filled with the pad character.

*Leading Bytes in Numeric Fields*

In NSL, numeric fields stored as bytes will always have the unused leading positions filled with spaces (ASCII Code 40B). Cobol uses zeroes (ASCII Code 60B). Therefore in some instances it may be necessary to convert COBOL to NSL on entry and NSL back to COBOL on exit.

A conversion routine has been designed to accomplish this. On entry to NSL, to subroutines using a record as a parameter, leading zeroes in numeric fields are converted to spaces and conversely on exit, leading spaces are converted to zeroes. This routine on entry also converts a trailing + sign to space and on exit a trailing space to a + sign.

To activate this conversion function you must set the Cobol flag, bit 15, in ITERM(3). This is accomplished by setting IARR(3) to −32768 plus the escape character, before calling COSCRI. See section 6.7.4.1.

Note that in NSHS because the sign byte is always trailing, COBOL numeric variables should be defined with 'SIGN TRAILING SEPARATE' clause.

## *Program example 5: COB-EX1 (C)*

This Cobol program demonstrates a few of the most important calls in NSHS. A picture, INV is called, and the operator may write data to the fields. The escape character (which is used to get out of the input mode) is here CTRL/@ which gives the numeric value 0 for the right byte of ITERM(3). See section 6.8.4.1. The corresponding value for the whole word is -32768 as used in the program for ITERM(3).

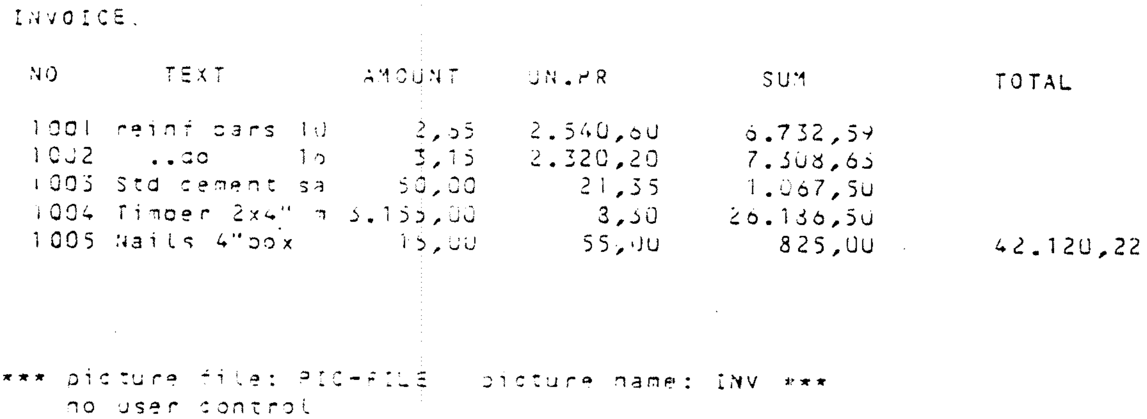
In a Cobol program, numerics must be represented by formal parameters. The values are set by the VALUE clause in the working storage section or in the procedure division. If special numeric values are used frequently in the program, the corresponding parameters are named accordingly, thus W-0 for 0, W-1 for 1 etc. See for example the call 'WRPTF' on line 94. The disadvantage is, that it is not possible to see the proper names of the parameters.

The data block to be loaded in front of the program is the same as used for the other programs. The proper values are, however, set from the program by the routine COSCR1 at the start of the program.

In this program the names of the subroutines are used to trace the processing of the program by means of the STOP statement efter each call. We might have used the DISPLAY statement, but sometimes the processing is so fast that it is not possible to see the text on the screen until it is overwritten by new text. The statement STOP (corresponding to PAUSE in a Fortran program) may then be used in the program to make a temporary stop in the processing. A 'CR' will make the program proceed. It is possible to give messages to the operator on the last line of the screen by means of the call 'WRMSGC'. In order to utilize the whole line, the MESSAGE should be defined with 79 characters (here only 12), and the text should be terminated by '*'. See line 100-101 in the compiling list.

```
NORD-10/100 COBOL COMPILER - 10176 A
SOURCE FILE:  COB-EX
OBJECT FILE: COB-EX

IDENTIFICATION DIVISION.
PROGRAM-ID.
          COB-EX1.
*The program demonstrates the most used calls in NSHS, calling
*one of the pictures in the picture file. The processing is
*traced by displaying the full name of the subroutine after
*each call. The program uses the picture INV on PIC-FILE:OBJ.

DATA DIVISION.
WORKING-STORAGE SECTION.
       77       TER-CHA                      COMP.
       77       NO-OF-FI                     COMP.
       77       STAT                         COMP.
       77       NO-FI-RE                     COMP.
       77       ERR-COD                      COMP.
       77       CURS-POS                     COMP.

       77       W-0                          COMP                VALUE  0.
       77       W-1                          COMP                VALUE  1.
       77       W-2                          COMP                VALUE  2.
       77       W-3                          COMP                VALUE  3.
       77       W-4                          COMP                VALUE  4.
    77    W-5                          COMP                      VALUE  5.
    77    W-6                          COMP                      VALUE  6.
    77    W-7                          COMP                      VALUE  7.
    77    W-8                          COMP                      VALUE  8.
    77    FTNO                         COMP                      VALUE 20.

    01    DAT-EL-IX-AR                 PIC X(52)         VALUE LOW-VALUE.
    01    FI-NO-AR                     PIC  X(40).
    01    PIC-NO-AR                    PIC X(16).
    01    SCR-REC                      PIC X(256).
    01    PIC-FI-NAM                   PIC X(81).
    01    PIC-NAM-STR                  PIC X(64).
    01    DEVICE                       PIC X(8).
    01    MESSAGE                      PIC X(12).

    01    T-INITA.
          03 INITA                     COMP          OCCURS  8.

PROCEDURE DIVISION.
MAIN-PROG SECTION.
P-START.

      MOVE 1      TO INITA(1).
      MOVE 3      TO INITA(2).
      MOVE -32768 TO INITA(3).
      MOVE 0      TO INITA(4) INITA(5) INITA(6) INITA(7).
      MOVE 8      TO INITA(8).

MOVE 'PIC-FILE:OBJ' TO PIC-FI-NAM.
MOVE 'ORD       INV       PER       ' TO PIC-NAM-STR.

CALL 'COSCR1' USING T-INITA.
STOP 'COSCR1 EXECUTED'.

CALL 'GTPICC' USING
  PIC-FI-NAM W-3 PIC-NAM-STR PIC-NO-AR STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'GET-PIC OK'.

CALL 'CLSCR' USING W-0 W-0 W-0 W-0 STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'CLEAR SCREEN (1) OK'.

CALL 'GTFDN' USING W-2 W-1 W-0 FI-NO-AR
  NO-OF-FI STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'GET-FIELD-NUMBERS OK'.

CALL 'WRPTD' USING W-2 STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'WRITE-PICTURE-TO-DISPLAY OK'.

CALL 'CLBUF' USING
  W-2 NO-OF-FI FI-NO-AR SCR-REC DAT-EL-IX-AR STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'CLEAR-BUFFER OK'.

CALL 'RFLDS' USING W-0 W-2 NO-OF-FI FI-NO-AR
  SCR-REC DAT-EL-IX-AR NO-FI-RE TER-CHA STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'READ-FIELDS OK'.

CALL 'CLSCR' USING W-1 W-1 W-8 W-0 STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'THE SCREEN PARTLY CLEARED'.

MOVE 'P-T' TO DEVICE.
CALL 'OPENFC' USING DEVICE W-0 FTNO STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'OPEN-FILE-COBOL OK'.

CALL 'WRPTF' USING FTNO W-1 W-0 W-2 NO-OF-FI
  FI-NO-AR SCR-REC DAT-EL-IX-AR STAT.
IF STAT NOT=0 GO TO ERR-PRO.
STOP 'WRITE-PICTURE-TO FILE OK'.

CALL 'CLOSEC' USING FTNO.
```

```
         MOVE 'YOU DID IT!'' TO MESSAGE.
         CALL 'WMSGEC' USING MESSAGE.

         GO TO FIN.

     ERR-PRO.
         CALL 'ERRORC' USING ERR-COD CURS-POS.
         DISPLAY ERR-COD CURS-POS.

     FIN.
         CALL 'ZBELL'
         STOP RUN.
```



```
INVOICE.

 NO        TEXT           AMOUNT     UN.PR        SUM         TOTAL

 1001 reinf bars 10        2,65   2.540,60     6.732,59
 1002   ..do     16        3,15   2.320,20     7.308,63
 1003 Std cement sa       50,00      21,35     1.067,50
 1004 Timber 2x4" m  5.155,00       3,50    26.136,50
 1005 Nails 4"box         15,00      55,00       825,00          42.120,22


*** picture file: PIC-FILE   picture name: INV ***
    no user control
```

*Figure H-1: Record written out by the command WRPTF.*

# APPENDIX I

# SINTRAN FEATURES

*Break and Echo Strategy*

SINTRAN-III contains features which, make character input/output to NSHS more efficient. These enhancements allow you to choose between two break and echo strategies. By setting the strategy in ITERM(8) to 1, the system upon entering a field, will select a strategy table that, even though it causes the terminal driver to echo legal characters, *will not generate* a break until the last character (maximum for the field) has been echoed. Should an illegal character be entered, it will not be echoed. Instead, a break will be generated and NSL will switch back to the normal strategy, to option 0, whereby for the rest of the field break and echo occurs on every character.

When ITERM(8) = 0, the following break and echo tables are operative:

BRK0:<br>
ECH0:

When ITERM(8) = 1,table 3 or 4 or 5 is used, depending on the edit code of the field being read.

BRK3: % No break on 1 or 9<br>
BRK4: % No break on letters<br>
BRK5: % No break on alphanumeric

ECH3: % echo 1—9<br>
ECH4: % echo letters<br>
ECH5: % echo alphanumeric

For further information see "SINTRAN-III Reference Manual" (ND-60.128.02). The tables are found in the SINTRAN-listing.

# APPENDIX J

# PUBLIC PICTURES

The following description requires some knowledge of Sintran and loaders. The readers not familiar with the subject are referred to the relevant manuals and ND's course department.

In a multiuser environment of some size, Public pictures will probably be used in paralell with Privat pictures.

There is no principle difference between Public and Privat pictures in the way they are created. But the Public picture file is implemented in the file system in a different way in order to utilize special Sintran features for fast and efficient access.

Public pictures should be used in following cases:

1) In reentrant program systems where a few pictures are used frequently by many background processes at the same time, for ex Menu pictures for request systems.

2) For real-time programming where a few pictures are often used by many terminals.

When Public pictures are used, both file access time and memory space are reduced.

# APPENDIX K

It is now possible by means of the program

SCREEN—COPY—FUNCTION

to copy pictures from one picture file to another. This program is delivered on the NSHS-diskett named ND-10013I-PART2:FLOPPY-USER.

The following text is identical to the program description for the product (SUT-2446A).

LOADING/OPERATING PROCEDURE,

```
    @PLACE-BIN    <input-file>
    @DUMP  "SCREEN-COP-2446A:PROG",  000000,  000000
```

AVAILABLE COMMANDS IN SCREEN-COPY-FUNCTION:

HELP

Help function: lists the commands

EXIT

Exit from subsystem

DEFINE-FILES

Define and open source and destination files. Operator is asked to give names of source and destination picture files. Instead of a source or destination name — if the previously defined name is to be re-used — one may simply type '*'.

If destination file is not a picture file, operator is asked whether it shall be made a picture file.

In case of an open file error, the file system error number is given.

MOVE PICTURES

Move picture(s) from source to destination file.
Operator is asked to give name of source-picture and destination-picture.
When copy is complete, another set of source and destination picture names may be entered, and so on until an empty source name (just CR) finishes the sequence and returns to command level.

Possible error messages are:

On source picture names:
NO SUCH PICTURE (on source file)

On destination picture names:
PREVIOUSLY USED (a picture with this name is already on dest. file)
FAILED, FILE TOO SMALL
TOO MANY PICTURES (a picture file may have max. 49 pictures)

LIST-PICTURES

List names of pictures on source and destination files.

DELETE-PICTURES

Delete picture(s) from destination file.

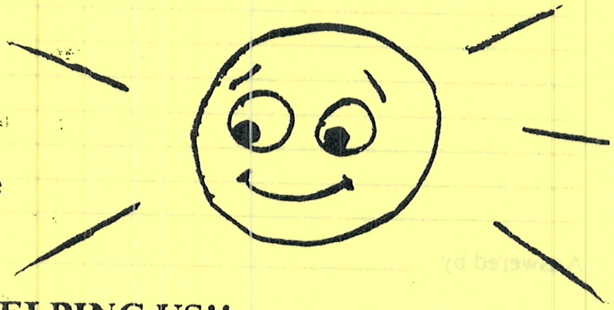
The SCREEN-COPY-FUNCTION is not a reentrant system.

## * * * * * * * * * * SEND US YOUR COMMENTS!!! * * * * * * * * * *

Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card - and an answer to your comments.

Please let us know if you
- find errors
- cannot understand information
- cannot find information
- find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!!



## * * * * * * * * * HELP YOURSELF BY HELPING US!! * * * * * * * * * *

Manual name:

Manual number:

What problems do you have? (use extra pages if needed)

Do you have suggestions for improving this manual?

Your name:

Date:

Company:

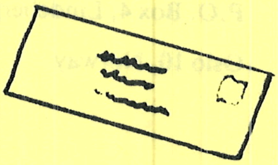
Position:

Address:

What are you using this manual for?

Send to:

Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway



Norsk Data's answer will be found on reverse side

Answer from Norsk Data ______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

Answered by ____________ ____________ ____________ ____________ Date ____________

--------
I I
I I
I I
--------

Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway

**Systems that put people first**

NORSK DATA A.S JERIKOVN. 20 P.O. BOX 4 LINDEBERG GÅRD OSLO 10 NORWAY
TEL.: 02 - 30 90 30 - TELEX: 18661