


Norsk Data



FOCUS **Screen Handling System** **Reference Manual**

ND-60.137.5 EN



FOCUS

Screen Handling System

Reference Manual

ND-60.137.5 EN

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. assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

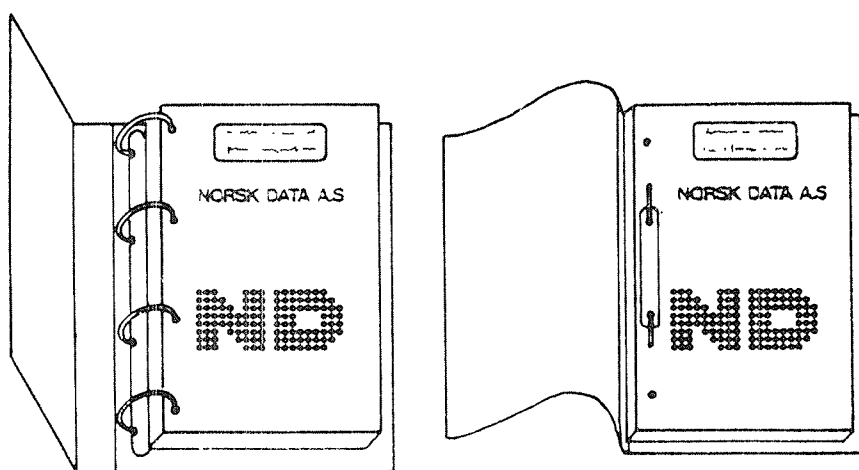
The information described in this document is protected by copyright. It may not be photocopied, reproduced or translated without the prior consent of Norsk Data A.S.

Copyright ©1985 by Norsk Data A.S.

This manual is in loose-leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

The loose-leaf form also allows you to place the manual in a ring binder (A) for greater protection and convenience of use. Ring binders with 4 rings corresponding to the holes in the manual may be ordered in two widths, 30 mm and 40 mm. Use the order form below.

The manual may also be placed in a plastic cover (B). This cover is more suitable for manuals of less than 100 pages than for large manuals. Plastic covers may also be ordered below.



A: Ring Binder

B: Plastic Cover

Please send your order to the local ND office or (in Norway) to:

Norsk Data A.S
Graphic Center
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

ORDER FORM

I would like to order

..... Ring Binders, 30 mm, at nkr 20,- per binder

..... Ring Binders, 40 mm, at nkr 25,- per binder

..... Plastic Covers at nkr 10,- per cover

Name

Company

Address

.....

City

Publ.No. ND-60.137.5 EN



Norsk Data A.S
Graphic Center
P.O.Box 25, Bogerud
0621 Oslo 6, 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 Customer Support Information (CSI) 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 and comments should be sent to:

Documentation Department
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Requests for documentation should be sent to the local ND office or (in Norway) to:

Graphic Center
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Preface:THE PRODUCT

This manual describes the FOCUS Screen Handling System. It is registered in the ND Software Library with the ND-numbers 210188E and 210341E.

THE READER

This manual should be of interest to anybody supervising data processing with FOCUS and particularly to programmers writing application programs that use the system. The first chapter - Introduction to FOCUS - is intended for those wanting a brief description of the system and how it works. There is also a special chapter for the terminal user and the supervisor of the system.

PREREQUISITE KNOWLEDGE

Application programs may be written in COBOL, FORTRAN or PLANC. The reader should have ample experience in the relevant language. We have also assumed that the reader is reasonably familiar with:

- the operating system SINTRAN
- one of ND's screen editors (PED, NOTIS-WP)

THE MANUAL

The present manual consists of two main parts. The first part (chapters 2 to 4) is dedicated to defining and maintaining forms (in NSHS called pictures). The second part describes the calls (from the FOCUS Library) to be used in application programs, and how to use these calls.

The appendixes contain a list of error messages, terminal information, changes from older FOCUS versions, loading FOCUS applications, examples of FORTRAN programs using FOCUS, and a summary of the PLANC interface to FOCUS. Enclosed is the "NOTIS reference card for non-NOTIS terminals" (ND-63.031Q.01), which gives non-NOTIS equivalents for the NOTIS keys.

RELATED MANUALS

ND FORTRAN Reference Manual	ND-60.145
COBOL Reference Manual	ND-60.144
SINTRAN III Introduction	ND-60.125
SINTRAN III Timesharing/Batch Guide	ND-60.132
PED User's Guide	ND-60.121
Introduction to NOTIS-WP	ND-63.001

T A B L E O F C O N T E N T S

<u>Section</u>	<u>Page</u>
1 Introduction to FOCUS - the Concept of the System	1
2 FOCUS-DEF - General	5
2.1 FOCUS-DEF Introduction	7
2.2 The Help Function in FOCUS-DEF	7
2.3 A Simple FOCUS-DEF Example	9
3 FOCUS-DEF Commands	15
3.1 Command Summary	17
3.2 Command Types	19
3.2.1 Commands to Enter Questionnaires	19
3.2.2 Direct Commands	19
3.2.3 Form File Commands	20
3.2.4 Utility Commands	22
3.2.5 The Environment Specification	24
3.2.5.1 Environment Commands	25
3.2.6 GROUPS	27
3.2.7 Executing SINTRAN Commands from FOCUS-DEF	28
4 Defining the Forms	29
4.1 Text Editing	31
4.1.1 Editing Keys	31
4.1.2 Using Display Attributes	33
4.1.3 The Alternative Character Set	35
4.2 Defining Fields	36
4.3 Field Types	39
4.3.1 The Field Format Syntax	39
4.3.2 Defining Other Field Attributes	41
4.3.3 Storage Types	42
4.4 Modifying Fields	43
4.5 Moving, Copying and Deleting Fields	43
4.5.1 Occurrences	44

Section	Page
4.6 Defining Form Extent	44
5 The RUNTIME Library	47
5.1 Introduction	49
5.2 Terms Used in This Chapter	51
5.3 Using FOCUS from COBOL, FORTRAN and PLANC	54
5.4 Initiation and Definition Calls	55
5.4.1 Initiate / Terminate FOCUS FCINITE	56
5.4.2 Declare Form File FCDECF	58
5.4.3 Declare Form Name FCDECFN	58
5.4.4 Declare Record FCDECRC	59
5.4.5 Set Display Attributes FCSETAT	60
5.4.6 Set Must Read FCSETMR	61
5.4.7 Clear Must Read FCCLMR	62
5.4.8 Define Next Field to be Edited FCNXFLD	62
5.4.9 Define Edit Start Function FCESFNC	63
5.4.10 Position Form FCPOSFO	64
5.5 Read / Modify Field(s)	65
5.5.1 Edit Record FCEREC	67
5.5.2 Edit Subrecord FCESUB	67
5.5.3 Edit One Field FCEFLD	68
5.6 Display Field(s)	70
5.6.1 Write Record FCWREC	71
5.6.2 Write Subrecord FCWSUB	71
5.6.3 Write One Field FCWFLD	72
5.7 Write Document to File	73
5.7.1 Open file FCOPEN	73
5.7.2 Print document on file FCPRDOC	74
5.7.3 Close File FCCLOSE	74
5.8 Message calls	75
5.8.1 Send Message FCZMSGE	75
5.8.2 Get Message FCGMSGE	76
5.9 Define Terminal Operator Interface	77
5.9.1 Break and Break Functions	77
5.9.2 Break Functions	77
5.9.2.1 Return to User Program Functions	78
5.9.2.2 Functions for Navigation in a Form	78
5.9.2.3 Other Functions	80
5.9.2.4 User Defined Functions	82
5.9.3 Define Function Keys FCCHRBR	84
5.9.4 Define Leaving Field Functions FCFLDBR	85
5.9.5 Define Leaving Form Functions FCFRMBR	86

Section	Page
5.10 Information / Status Calls	87
5.10.1 Define Edit Status Buffer FCDESB	87
5.10.2 Get Edit Status FCEDSTA	89
5.10.3 Get Screen Information FCSCRIN	90
5.10.4 Get Form Information FCFRMIN	90
5.10.5 Get Field Information FCFLDIN	91
5.10.6 Get Field Name FCFLDNA	92
5.10.7 Get Field Status FCFLDST	92
5.11 Multi Forms Handling	94
5.11.1 Save Form FCSAVFO	94
5.11.2 Clear Form FCCLFO	95
5.12 Other Form Related Calls	96
5.12.1 Clear Data Record FCCLREC	96
5.12.2 Clear Subrecord FCCLSUB	97
5.12.3 Write Dots in Fields FCWDOTS	98
5.12.4 Clear Fields FCCLFDS	98
5.12.5 Get Data Elements from a Data Record FCGSUB	99
5.12.6 Put Data Elements into a Data Record FCPSUB	100
5.12.7 Get Form Line FCGLINE	100
5.12.8 Write Leading Texts FCWLTX	101
5.12.9 Redisplay Form FCRDFO	102
5.13 Form Independent Calls	103
5.13.1 Read Text FCRTXT	103
5.13.2 Write Text FCWTXT	104
5.13.3 Clear Rectangular Area on Screen FCCLSCR	105
5.13.4 Position Cursor FCPCUR	105
5.13.5 Empty Buffer FCEBUF	106
5.13.6 Read Character FCRCHR	106
5.13.7 Bell FCBELL	106
6 Terminal Operator Interface	107
6.1 Editing a Field	109
6.2 Navigating in a Form	111
6.3 Terminating Form Editing	112
6.4 Other Functions	113
7 Loaded Forms	115
8 User Defined Control	119

<u>Section</u>	<u>Page</u>
 <u>APPENDIX</u>	
A	Error Codes from FOCUS-RTS 123
B	Terminal Information 127
C	Converting from Older FOCUS Versions to the G Version 131
D	Loading FOCUS Applications 137
E	A Few Examples of Application Programs Using FOCUS 143
F	PLANC Interface to FOCUS-LIB 157
 Index	 163

CHAPTER 1

INTRODUCTION TO FOCUS - THE CONCEPT OF THE SYSTEM

1 INTRODUCTION TO FOCUS - THE CONCEPT OF THE SYSTEM

The use of FOCUS is based on two basic elements, the forms (defined on the screen) and the application programs which use the forms.

The FOCUS system may be used for entering data to and/or displaying data from record files or data bases via the terminal.

The Form is the basic element. A form consists of permanent text which cannot be modified at runtime, and data fields. The permanent text is often called 'leading text', but text and data fields may occur in any order.

Person register	FOCUS demo form
Name	: Nils-Petter Nilsen
Address	: Stensgt 14.....
Birth date	:
Sex	: .
Employee no	:
Salary	:

Fig. 1. A form for Person records

In Fig. 1 is shown a form for registration of person records, as it appears on the screen. The first two fields have been filled in, the others are still empty. The empty data fields are indicated by dots.

When the form is created, it is possible to define automatic data control on any field. If the data entered by the user of the application program is not acceptable because of the type or value of the field, the data will be rejected with a "beep", and the user may try again with correct data.

The leading texts can be created with display options like 'underlined', 'inverse video' and 'blinking' using the display attribute specification facility.

When a field (in a form) is created, help forms with information about the current field can be defined. These help forms are called at runtime when the HELP key is used.

CHAPTER 2

FOCUS-DEF - GENERAL

2 FOCUS-DEF - GENERAL

2.1 FOCUS-DEF INTRODUCTION

FOCUS-DEF is used interactively from the terminal. Forms are defined and modified in almost the same way as when a text editor is used to write and edit a document.

Forms are stored on special form files after they have been defined. These files are contiguous :FORM files in SINTRAN III. They have a special format and are created from FOCUS-DEF.

FOCUS-DEF places no limits on how many form files there can be. The number of forms that can be stored on a form file is only limited by the size of the file.

Each form's area is limited by the size of the screen. The two bottom lines are reserved for use by FOCUS-DEF. The largest possible line width is 80 characters. A form consists of leading texts and fields. You may define a maximum of 31 fields per line.

2.2 THE HELP FUNCTION IN FOCUS-DEF

There are two types of help information available in FOCUS-DEF:

When command parameters are prompted,
and inside definition questionnaires,
The HELP key can be used to get
information relevant to the current
situation.

FIELD DEFINITION

Field name : Field format :

~~~~~ To specify more field attributes use the ↓ (down arrow) key. ~~~~~  
~~~~~

There is also a more general help information that you get when you press the HELP key when the cursor is on the command line or in the form definition area of the screen. The main help picture looks like this:

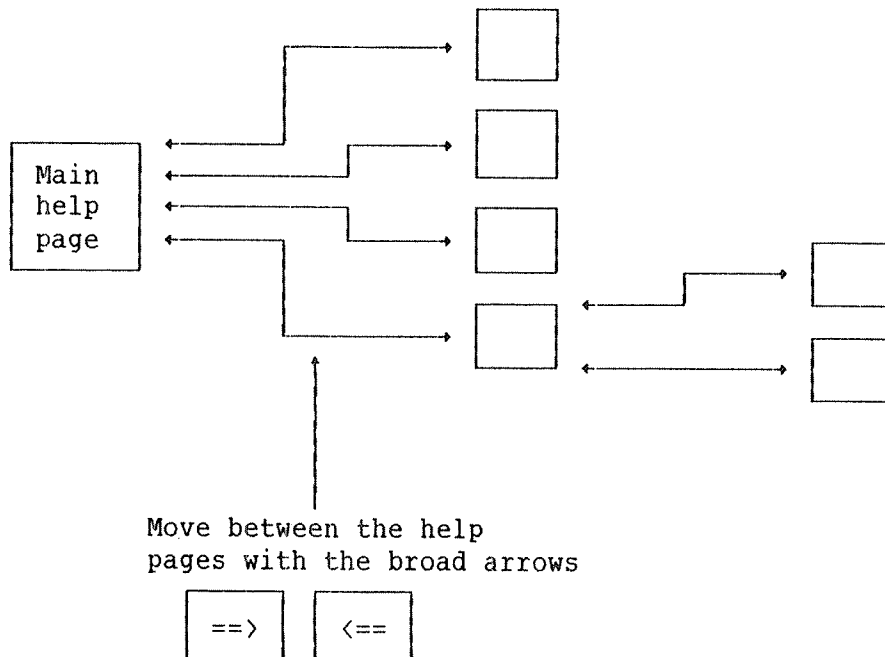
● F O C U S - D E F H E L P ●

- List of All Commands
- General Information
- Form File Commands
- Definition Commands
- Other Commands
- Edit and Definition Functions

Navigate to the desired topic and press or .

Use the key to get back to the previous help picture.

This general help information has a tree structure which is shown schematically in the following diagram (the actual help information has more pages):



2.3 A SIMPLE FOCUS-DEF EXAMPLE

This section explains step by step how to define a simple form. It is intended

- 1) as an introductory example to give new users an impression of how FOCUS-DEF is used, and
- 2) to show the use of different types of commands in FOCUS-DEF. (This is explained more fully in the first sections of the next chapter).

Enter FOCUS-DEF by typing "FOCUS-DEF" after the SINTRAN III prompt. The cursor will be positioned in the lower left corner of the screen, on the command line. Move the cursor into the screen picture by pressing the \ (HOME) key.

In this example we will define a very simple form to enter or display person records:

Name:
Address:
Sex:	

The cursor is now positioned in the upper left corner of the screen. Position it where the leading text "Name:" is to begin (i.e., a few lines lower and a few character positions further to the right). Enter the text "Name:" by simply typing it in. Then, move the cursor further to the right, to where the first field is to begin (as shown above), and press the FIELD key.

A questionnaire will be displayed, and can be filled in as shown below:

1) Enter the field name.
You may call it whatever you like, just remember that this is the name used to refer to the field in the calls from the application program.
Use the ← key to move to the next field in the questionnaire.

2) Enter the field format. This is a COBOL PICTURE like character string. <1>
Use the ← key to leave the questionnaire.

FIELD DEFINITION

Field name : PNAME Field format : X(35)←

To specify more field attributes use the ↓ (down arrow) key.

The field extent will be shown on the screen as a line of dots:

Name:

Define the next field, "Address", in the same way as the name field (two lines below), but with the last one, "Sex", we want to restrict the values that may be entered in the field:

FIELD DEFINITION

Field name : SEX Field format: X

To specify more field attributes use the ↓ (down arrow) key.

When you get to the field format, use the ↓ key instead of ← this time.

<1> See the separate section on the field format.

The questionnaire will expand. Move through it by using the \leftarrow key until you get to "Legal values":

All additional fields in the expanded questionnaire are optional. Some of them will remain empty if you do not fill them in.

Some fields, however, have default values that will appear when you press the \leftarrow key.

FIELD DEFINITION

Field name	: SEX	Field format	: X	
Storage type	:	Legal characters	: X	\leftarrow
Justification	: None	Blank when zero	: No	\leftarrow
Help form	: \leftarrow	Occurrence range	: Line	\leftarrow

Field functions before editing:

- Initial value :
- FCUCONT par. :


Field functions after editing:

- Default value :
- System routine :
- Legal values : 'M', 'F' \leftarrow
- Illegal values :
- FCUCONT par. :

"Legal values" means that all values except these two will be rejected when entered in this field at runtime. Apostrophes must be used to enclose non-numeric values specified in the questionnaire.

When you have finished your specifications / changes, you may use the

FUNK+ \leftarrow

finish function, the  key, to bring you out of the questionnaire.


After you have finished defining the form, you can write it to a form file. To create a new form file, go back to the command line with the \backslash key, and give the command

FD>CREATE-FILE

Form files cannot be created using the SINTRAN command CREATE-FILE!

You will enter the following command form:

```
##### C R E A T E - F I L E #####
#
# Form file   : EXAMPLES
# Size       : 20
# Environment : FOCUS-ENV
#####
```

Here, too, the default values
will appear when you use the  key.
(The environment specification is
explained on page 24).

Alternatively, the command and the parameters can all be written on
one line:

FD>CREATE-FILE EXAMPLES,,,

The commas will cause the current size and the current environment
file to be filled in as default values.

To store the form definition on the file, use the command

FD>WRITE-FORM

You will enter the following command form:

```
##### W R I T E - F O R M #####
#
# Form file : EXAMPLES
# Form      : "PERSONS"
#####
```

The name of a new
form must be enclosed
in quotes.

The file name is default,
since it has been used
earlier.

This command may also be given on one line, with its parameters:

FD>WRITE-FORM , "PERSONS"

The comma will cause the current form file name to be filled in as
default value.

To exit from FOCUS-DEF, use the EXIT key or the EXIT command.

CHAPTER 3

FOCUS-DEF COMMANDS

3 FOCUS-DEF COMMANDS

3.1 COMMAND SUMMARY

This section, a printout of three of the help pictures in FOCUS-DEF, lists all commands in the system. The rest of the chapter will give a more detailed description of the commands.

FORM FILE Commands

READ-FORM <form file> <form>	● Get a form from file
WRITE-FORM <form file> <form>	● Store form on file
DELETE-FORM <form file> <form>	● Delete form on file
DESCRIBE-FORMS <form file> <form> <output file>	● Give form information
LIST-FORMS <form file> <form> <output file>	● List forms in form file
LIST-FILES <form file> <output file>	● List form files
CREATE-FILE <form file> <size> <environment file>	● Make new form file
ENLARGE-FILE <form file> <extra size>	● Enlarge a form file
GET-ENVIRONMENT <form file>	● Get/modify environment specification
SAVE-ENVIRONMENT <form file>	● Save environment specification
MAKE-UP-TO-DATE-FORMS <form file>	● Update forms after environment change

Parameters to these commands may be supplied directly on the command line. Defaults are obtained by omitting a parameter (i.e., indicated by a comma). If no parameters are supplied, you will enter a command prompting form.

D E F I N I T I O N C o m m a n d s



(home) ● Enter form definition/modification

CLEAR-DEFINITION ● Clear current form definition
GET-KEY-VALUE ● Show internal code for any key
ENVIRONMENT ● Inspect/modify "environment"
GROUP ● Define/modify/delete field group
LIST-GROUPS <group name> ● List defined field groups

Commands without parameters will bring you into a questionnaire
if additional information/specification is required.

O t h e r C o m m a n d s

HELP / HELP ● Give help information

EXIT / EXIT ● Exit from FOCUS-DEF

FUNC + @ ● Redisplay screen

@ "XXX" ● Execute the SINTRAN command "XXX"

3.2 COMMAND TYPES

Commands are entered on the command line, and may be truncated provided there is no ambiguity. Some of the commands have no parameters, and are only used to enter a questionnaire. Other commands, called direct commands, have parameters that may be specified together with the direct command on the command line, or in a command form that will be entered if parameters are missing or erroneous.

3.2.1 COMMANDS TO ENTER QUESTIONNAIRES

Some of the commands in FOCUS-DEF are used to enter a questionnaire. Each questionnaire consists of several parameter fields, with prompts telling you what to fill in. In any parameter field the HELP key may be used to get relevant help information on the screen. Pressing any key brings you back to the questionnaire.

Most parameters have default values. If you press `↵` in an empty parameter field, the default value will be filled in. If nothing is filled in and the `↵` key brings you to the next parameter field, the parameter is optional.

Pressing the CANCEL key or the EXIT key inside a command form or questionnaire will abort the command, i.e., the situation will be as immediately before you entered the questionnaire.

3.2.2 DIRECT COMMANDS

These commands differ from the questionnaire commands in the following way:

One or more of the parameters may be specified together with the command on the command line.

- If all parameters are correctly given, the command will be executed immediately.
- If illegal parameters are given, or if some parameters are missing, a command form is entered where the parameters are prompted for.

Default parameters may be specified on the command line with commas.

Example:

After having read a form definition from the file and modified it, you may write it back on the form file using the `WRITE-FORM` command with default parameters:

`FD>WRITE-FORM , ,`

The commas will cause the current form file name and the current form name to be filled in as default values.

3.2.3 FORM FILE COMMANDS

FOCUS form definitions are stored in form files. Several definitions may be stored in the same file, depending on the file size.

To use the form file system, the following commands are available:

CREATE-FILE - Create a new form file

This command creates and initializes a form file.

NOTE: this is a command in FOCUS-DEF. Form files cannot be created by using the SINTRAN command CREATE-FILE.

The parameters are:

- form file name
- form file size (in pages, with a default of 20)
- environment source file

The environment specification in the environment source file will be used as the new form file's environment (see p. 24). Default file name is `FOCUS-ENV`, which is the name of the original environment source form file delivered with FOCUS.

ENLARGE-FILE - Enlarge a (full) form file

If a form file becomes full, it is possible to expand it so that it can hold more forms.

The parameters are:

- form file name
- number of extra pages

LIST-FILES - List form files

One SINTRAN user may have several FOCUS-DEF form files. To get a list of your form files, use the LIST-FILES command. Default output file is TERMINAL.

The list is displayed as in the following example:

Form file	Forms in file	Pages in file	File space used
NEW-FORM-FILE	7	18	47 %
STAR-FILE	11	22	86 %
EXTRA-FILE	41	100	34 %

WRITE-FORM - Write a defined/edited form to a form file

READ-FORM - Read a form from a form file

DELETE-FORM - Delete a form on a form file

These three commands enable you to maintain your form file.

All three commands take two parameters:

- form file name
- form name

To write a new form to the form file, you have to enclose its name in double quotes, for example, "MY-NEW-FORM".

To modify an existing form, read it from the form file using the READ-FORM command. Then modify it as you want, and re-write the modified version using WRITE-FORM command.

NOTE: the modified version will over-write the old one!

When re-writing a modified form, you can use the default WRITE-FORM parameters. Press the ← key in empty parameter fields to look at the default values!

To modify a form and still keep the old version, write the modified version using a new name in double quotes, or take a backup copy of the old form before starting to modify it.

When entering FOCUS-DEF an automatic READ operation is available. Start FOCUS-DEF the following way:

@FOCUS-DEF FORMS-3 MY-FORM

will cause FOCUS-DEF to READ the form MY-FORM from the form file FORMS-3 as a part of the start-up procedure. The MY-FORM definition will appear on your screen immediately, ready for inspection or modification.

LIST-FORMS - List forms stored in a form file

This command will list forms stored in a given form file.

The parameters are the same as for READ/WRITE commands. An abbreviated form name may be entered, and only the forms with a matching name will be listed.

The LIST-FORMS command gives the following format:

Form	Size
DUPLIC	1254
PER	1422

The size given here is the figure used to compute the size of the form buffer in FOCUS RUNTIME SYSTEM (FOCUS-RTS).

3.2.4 UTILITY COMMANDS

CLEAR-DEFINITION - Clear current form

If you have finished your work on one form and want to start a new one, the CLEAR-DEFINITION command should be used. This command clears the current form from the screen, and initializes the program to start a new form definition.

If you have not written the previous form to the form file (using WRITE-FORM), a warning will be issued. You may write your form and then use the CLEAR-DEFINITION command once more. If you do not want to save the form, just repeat the CLEAR-DEFINITION command.

DESCRIBE-FORMS - Get information about current form

This command is useful to get a thorough description of your form. Default output file is **TERMINAL**.

Example:

You want a full documentation of the form **PERSONS** in the form file **EXAMPLES**, so that you can check the form attributes on the printout. If you want output on the line printer, enter:

FD>DESCRIBE-FORMS EXAMPLES PERSONS LINE PRINTER

A truncated form name may be entered, and only the forms with a matching name will be listed. If you want to truncate the form name you must give a "*" as the last character in the truncated name.

EXIT - Exit from FOCUS-DEF

The **EXIT** key may be used to exit from FOCUS-DEF, whether in command position or in edit mode. In command position the command **EXIT** may also be used.

If the current form definition and/or environment specification has not been written to a file after the last change, a warning message will be issued. If you want to exit without saving the form/environment specification, use the **EXIT** key once more. Alternatively, you may use any key to get back to command position, and use the **WRITE-FORM / SAVE-ENVIRONMENT** commands.

Pressing the **EXIT** key inside a command form or questionnaire will abort the command.

FUNC @ - Redisplay the screen

To redisplay the screen, use the **FUNC** key followed by the **@** character.

GET-KEY-VALUE - Show internal code for a key

This command is used in order to display the internal code generated for any key on the terminal keyboard. It is intended to be a helpful tool in connection with the definition of function keys in FOCUS-RTS.

3.2.5 THE ENVIRONMENT SPECIFICATION

An environment in FOCUS-DEF is a set of language dependent parameters, e.g., error messages and character set definitions, used by FOCUS-RTS.

Each form file contains an environment specification that may be retrieved and modified using FOCUS-DEF:

Environment specification
FORM-1
FORM-2
FORM-3
.....

When a form is written to a form file, the environment information needed for that particular form is taken from that form file's environment specification. This means that if the form file's environment specification is changed in the interval between writing two forms, the environment dependent parameters will be different for these two forms. It also means that if a form is read and written after a change in the form file environment, the form's environment dependent parameters will be changed.

3.2.5.1 ENVIRONMENT COMMANDS

GET-ENVIRONMENT - Get/modify environment specification

This command reads the environment specification from a specified form file and enters a questionnaire to start inspecting or modifying it. The original environment specification delivered with FOCUS looks like this:

```

##### E N V I R O N M E N T #####

Decimal point : .           Number group separator      : ,
Expand text   : Expand      Alternative character set text : Alt. set
Field indicator (dot) : .    Alphabetic range (lowercase) : #a:#z

Ordinary character sets:
A → #A:#Z,#a:#z,# ,#-
X → # :# ~
N → #0:#9,#. ,#+,#-
Special character sets:
S1 →
S2 →
S3 →
System messages:
→ Illegal field value!
→ Illegal date!
→ Illegal "personnumber"
→ Illegal "bankaccount" number
→ Illegal "postalaccount" number
#####

```

DECIMAL POINT is the character that is to be used as decimal point by FOCUS-RTS.

Examples: . (dot) , (comma)

NUMBER GROUP SEPARATOR is the character that is to be used as number group separator by FOCUS-RTS.

Examples: , (comma) . (dot) (space)

Examples of use (comma as separator): 1,000,000 50,000.00

EXPAND TEXT is the text to be displayed when in "expand" mode.

ALTERNATIVE CHARACTER SET TEXT is the text to be displayed when in "alternative character set" mode.

FIELD INDICATOR (dot) is the character that will be displayed in a field if the FCWDOTS call is used or when edit modus 1 is used in FCEREC, FCESUB or FCEFLD (see FOCUS-RTS).

ALPHABETIC RANGE defines the range of characters to be converted to uppercase letters when the system routine UC is used.

Under **ORDINARY CHARACTER SET** you may modify the definition of the three standard character sets to be referenced in field definitions:

- A (Alphabetic), X (alphanumeric) and N (Numeric).

Under **SPECIAL CHARACTER SETS** you may define your own character sets to be referenced in field definitions.

Syntax for character set definitions: #<ASCII character> or a range, (e.g., #A:#Z, #0:#9, #-).

A plus (+) indicates that characters from the alternative set are allowed.

SYSTEM MESSAGES are predefined error messages for the checking routines. (The reason for the quotes in the last three messages is that the checking routines that use them follow Norwegian standards.)

ENVIRONMENT - Inspect/modify environment

The environment specification stays in memory until you leave FOCUS-DEF, and can be modified at any time by using this command.

SAVE-ENVIRONMENT

This command writes the environment specification back to a form file.

MAKE-UP-TO-DATE-FORMS - Update forms

All form definitions on the form file will be changed to correspond to the file's current environment specification.

Example:

FD>MAKE-UP-TO-DATE-FORMS MYFILE

will cause all forms on MYFILE to be updated.

3.2.6 GROUPS

A field group definition may in principle be viewed as just a named character string. The "character string" must be a list of field names.

The group concept will enable you to make your programs more easily readable (and possibly shorter). You will also achieve a higher degree of independence between the form definition and the program(s) written to use the form, e.g., you may change the order in which to edit some fields without having to change the program at all.

NOTE that the fields in a field group may be defined with or without occurrence numbers. If no occurrence number is supplied, the occurrence number(s) in the FOCUS calls using the group definition will be used. This may be particularly useful when repeatedly editing several occurrences of the same set of fields, (e.g., SIBAS records.)

The (name of a) group definition may be used in the FOCUS-RTS calls wherever a field list is expected.

The command used to define a group is called:

GROUP - Define/modify/delete a field group

This command enters the following questionnaire:

```
##### G R O U P #####
```

Group name :

```
:  
:  
:  
:  
:  
:
```

```
#####
```

The command is used to DEFINE, MODIFY or DELETE a group:

- To DEFINE a new group: enter a new group name, and proceed to define the field group.
- To MODIFY a group: enter the group name and, proceed to modify when the group definition has been displayed.
- To DELETE a group: enter the group name and, delete the name when the group definition has been displayed.

A FIELD GROUP DEFINITION is a sequence of field specifications, separated by space or comma.

- a FIELD SPECIFICATION consists of a field name which may be followed by an occurrence part.
- the OCCURRENCE PART starts with a period (.) followed by an occurrence number or an occurrence range.
- an OCCURRENCE RANGE consists of a first and a last occurrence number separated by a colon (:).

Field specification examples: FIELD1.1:8, FIELD2, FIELD3, FIELD4.2

The command

LIST-GROUPS

will list all the groups defined for the current form.

3.2.7 EXECUTING SINTRAN COMMANDS FROM FOCUS-DEF

It is possible to execute SINTRAN commands without leaving FOCUS-DEF, by preceding the command with the character @, i.e.,

FD>@xxxx

where 'xxxx' is any legal SINTRAN command.

C H A P T E R 4

DEFINING THE FORMS

4 DEFINING THE FORMS

4.1 TEXT EDITING

Text editing in FOCUS-DEF is similar to PED/NOTIS-WP, but NOTE one important difference: marked text cannot be moved, copied or deleted. Text marking can only be used to set display attributes (p. 33). Marked fields, however, can be moved, copied and deleted (p. 43).

4.1.1 EDITING KEYS

The following editing keys and functions are available:

F1	● Delete line (without fields)
F2	● Insert blank line
CANCEL	● Undo editing on line, or ● Reset marking in text (or field), or ● Insert last deleted line (without fields)
EXP	● Set/reset expand mode
SHIFT+ F1	● Set/reset alternative character set

<F>+"x"	● Move cursor right to first "x"
<R>+"x"	● Move cursor left to first "x"
<C>	● Copy one character from previous line
<N>	● Copy one character from next line
<P>+"x"	● Copy previous line up to and including "x"
FUNC+<P>+"x"	● Copy next line up to and including "x"
<A> / X	● Delete character
<D>+"x"	● Delete characters up to and including "x"
<V>	● Give cursor position information

The notation
<X> means
the CTRL key
plus the
character X.

<D>+<D>	● Delete line
<L>	● Insert blank line
<W>	● Restore last deleted line
<E>	● Set/reset expand mode
FUNC+A	● Set/reset alternative character set

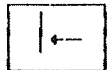
These functions
have their own
function keys.

SHIFT+ ==> ● Move cursor past last significant character on line

SHIFT+ <== ● Move cursor to first significant character on line



● Move cursor right to next tabulator



● Move cursor left to previous tabulator

FOCUS-DEF has
fixed tabulator
settings: every
fifth position.

<F><F>	● Move cursor past last significant character on line
<R><R>	● Move cursor to first significant character on line
<T> / <I>	● Move cursor right to next tabulator
<Y> / <U>	● Move cursor left to previous tabulator

4.1.2 USING DISPLAY ATTRIBUTES

(For a description of display attributes in fields, see p. 60.)

Display attributes (inverse video, high and low intensity, blinking, underlined, or invisible) can be achieved in one of two ways:

- 1) By first setting the desired attribute(s), and then entering the text, or
- 2) with existing text, by first marking the text, and then changing it from normal to the desired attribute(s).

The first method is as follows: let us say you want text in inverse video. First you set the display attribute to inverse video by pressing the following keys:

SHIFT +

aaa <u>aaa</u>

 and then

I

The second key specifies an attribute type, where:

N	means	Normal
I	..	Inverse video
H	..	High intensity
L	..	Low intensity
B	..	Blinking
U	..	Underlined
X	..	Invisible

So far no visible reaction can be observed. The inverse video feature is activated by pushing the key

aaa
<u>aaa</u>

and deactivated by pushing the key once more.

You can now activate the inverse video attribute by pushing the 'aaa' key, and start writing. The text written will appear in inverse video as the writing goes on. At the same time the message 'Inp' (i.e. normal input mode) on the line above the command line is exchanged with 'Atr' (display attribute mode), shown in inverse video.

To change attribute on existing text, you have to:

- 1) Set the desired attribute by pressing the SHIFT +

aaa
<u>aaa</u>

 and then specifying the attribute type as shown above.
- 2) Move the cursor to the text where you want to change attribute(s).
- 3) Mark the whole text by pressing the MARK key at the beginning and at the end of the text you want to change. (NOTE: in FOCUS-DEF marking is only possible within one line).
- 4) Press the

aaa
<u>aaa</u>

 key. The whole marked area will now get the current attribute setting.

Attributes are combined by using the SHIFT +

aaa
<u>aaa</u>

 repeatedly

e.g., to get low intensity inverse video:

SHIFT +

aaa
<u>aaa</u>

L

 and SHIFT +

aaa
<u>aaa</u>

I

How to reset attributes:

If you want to remove the attribute(s) from the text, you have to:

- 1) Press SHIFT +

aaa
<u>aaa</u>

 and then

N

- 2) Mark the whole text you want to change by pressing the MARK key at the beginning and at the end of the text.
- 3) Press the 'aaa' key, and the text's attribute will be reset to normal.

Display attributes are not available on all terminal types. See appendix B.

4.1.3 THE ALTERNATIVE CHARACTER SET

	<div>ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz</div> <div>ΑΒΓΔΕΦΓΗΙΪΚΛΜΝΟΠΘΡΣΤΥΦΧΨΖ αβξδεφγηιϿκλμνοπθρστυνωχψζ</div>	
Letters		
	<div>1 2 3 4 5 6 7 8 9 0 .</div> <div>⌈ ⊥ ⌋ ⊕ ⊖ ⊗ ⊘ ⊙ ⊚ ⊛ ⊜ ⊝ ⊞ ⊠ ⊡ ⊢ ⊣ ⊤ ⊥ ⊦ ⊧ ⊨ ⊩ ⊪ ⊫ ⊬ ⊭ ⊮ ⊯ ⊰ ⊱ ⊲ ⊳ ⊴ ⊵ ⊶ ⊷ ⊸ ⊹ ⊺ ⊻ ⊼ ⊽ ⊾ ⊿</div>	<div>! " # \$ % & ' () =</div> <div>␣ ␤ ␥ ␦ ␧ ␨ ␩ ␪ ␫ ␬ ␭ ␮ ␯ ␰ ␱ ␲ ␳ ␴ ␵ ␶ ␷ ␸ ␹ ␺ ␻ ␼ ␽ ␾ ␿</div>
Graphic		
and		
	<div>, ; < > : ?</div> <div>↓ ↑ ↔ → ↪ ↩ ↪ ↩ ↪ ↩</div>	<div>+ / - *</div> <div>• © ~ £</div>
special symbols		<div>æ ø å Æ Ø Å</div> <div>{ } [\]</div>

To set/reset the alternative character set mode use SHIFT + F1 .

The top lines show which key to use to get the character below.

4.2 DEFINING FIELDS

The first step in defining a field is optional: you may set the field length by marking the field extent (use the MARK key at the beginning and at the end of the field). Otherwise the length will be determined when you specify the field format (see below).

Then you position the cursor where you want the field to begin (if you have marked an area: anywhere inside the area), and press the FIELD key. The following questionnaire will appear:

```
=====
```

F I E L D D E F I N I T I O N

Field name : Field format :

```

===== To specify more field attributes use the ↓ (down arrow) key. =====
=====
```

The **FIELD NAME** may be up to 8 characters long. All alphanumeric characters and the underline character () may be used. The name must start with an alphabetic character.

As **FIELD FORMAT** you may enter:

- A format (COBOL picture like) for the field (see p. 39).
- The name of another field (start with =), to get its attributes copied.
- The name of a system field type (start with ^) to be used in this field.

Available system field types:

^date (Format: YYMMDD)

^personnumber ^bankaccount ^postalaccount

Numbers entered in fields of these types are checked by routines in FOCUS-RTS. (For ^personnumber, ^bankaccount and ^postalaccount, the checking routines follow Norwegian standards.)

If you use the $\leftarrow \downarrow$ key to leave the last field in this questionnaire, you will go right back to editing the form. If you use the \downarrow key, the questionnaire will expand:

=====

F I E L D D E F I N I T I O N

Field name	:	Field format	:
Storage type	:	Legal characters	:
Justification	:	Blank when zero	:
Help form	:	Occurrence range	:

Field functions before editing:

- Initial value :
- FCUCONT par. :

Field functions after editing:

- Default value :
- System routine :
- Legal values :
- Illegal values :
- FCUCONT par. :

=====

Possible **STORAGE TYPES** are (numeric fields only):

- ACD : Unpacked decimal (Corresponds to COBOL 'USAGE IS DISPLAY')
- BCD : BCD format (Packed decimal)
- Int2 : Integer value stored in 2 bytes
- Int4 : Integer value stored in 4 bytes
- Real4 : Floating point, stored in 4 bytes
- Real6 : Floating point, stored in 6 bytes
- Real8 : Floating point, stored in 8 bytes

HELP FORM is the name of the form to be displayed when the HELP key is pushed in this field.

As **LEGAL CHARACTERS** (non-numeric fields only) you may choose one of the predefined "character sets" (indicating which characters are to be accepted in the field).

These are: A (Alphabetic), X (Alphanumeric), N (Numeric),
and the "special" sets S1, S2 and S3.

Alternatively, you may enter an expression specifying the legal characters.

Syntax: #<ASCII character> or a character range (E.g., #-, #A:#Z, #0:#9)

A + (plus) indicates that characters from the alternative set are allowed.

JUSTIFICATION modes (non-numeric fields only) are: None, Left or Right.

If **BLANK WHEN ZERO** is used and the content of the field is zero (or blank if string-field), only blank characters will be displayed, that is, editing characters, decimal point etc. are ignored.

Under **OCCURRENCE RANGE** you may specify if the occurrence range should be LINE-oriented or COLUMN-oriented.

As **INITIAL VALUE** you may specify a value to be preset in the field before editing. It will appear when the field is entered.

Under **DEFAULT VALUE** you may specify a value to be entered in the field after editing, if the field is empty and carriage return (↵) is used to leave the field.

For non-numeric fields the initial/default value must be enclosed in apostrophes ('). You get today's date as a default by specifying \$DD.

Under **FCUCONT PAR** you may specify a string to be used as the first parameter in the user defined routine FCUCONT. If nothing is entered here, the routine will not be called. Trailing spaces will be stripped.

For a description of FCUCONT, see p. 121.

SYSTEM ROUTINE can be one of the routines mentioned under "field format". The field will be filled in automatically if one of the system field types are used as "field format", but if you want these routines with a field format entered manually, you may specify it here. Available routines are:

- Date, Personnumber, Bankaccount, Postalaccount

In addition, if the UC routine is specified, the field value will be converted to uppercase letters. (se also ALFABETIC RANGE in the ENVIRONMENT specification.)

Under **LEGAL VALUES** you may specify an expression defining values and/or ranges of values to be accepted in the field.

Syntax: <operator><operand> or <operand>:<operand>

Operators are: = > < >= <= > <
(If operator is omitted, = is assumed)

For non-numeric fields the operands must be enclosed in apostrophes (')

Under **ILLEGAL VALUES** you may specify an expression defining values and/or ranges of values NOT to be accepted in the field. Specifying '' as illegal value indicates, that the field has to be filled in, i.e. it will be illegal to type ↵ in an empty field.

4.3 FIELD TYPES

FOCUS supports four different field types:

- Numeric fields
- Numeric E-format fields
- String fields
- Text fields

When using numeric E-format fields, numeric values will be displayed with an exponent part.

Text fields are string fields with the additional editing functions word-wrap, insert line and delete line. They consist of a fixed number of lines, which are all occurrences of the same field. The order of the elements in the data record and the default edit order will differ from the standard when using text fields: the data elements for a text field will be adjacent in the data record, and all lines in a text field will be edited, in occurrence range order, before the next field in the form is entered.

4.3.1 THE FIELD FORMAT SYNTAX

For Numeric fields and String fields, the syntax for the field format specification is based on the rules for COBOL's PICTURE clause, with the following exceptions:

- The letters A, S and V may not be used
- The symbols CR, DB and CS may not be used
- The numeric symbol '9' may not be in alphanumeric formats

Short description of the legal format specification symbols:

- X denotes any alphanumeric character
- 9 is replaced by a numeric digit, leading zeroes as 0
- Z is replaced by a numeric digit, leading zeroes as space
- * is replaced by a numeric digit, leading zeroes as asterisk
- . marks the decimal point position (maximum one occurrence)
- , gives a comma at the corresponding position in the field
- / gives a slash at the corresponding position in the field
- B gives a space at the corresponding position in the field
- 0 gives a zero at the corresponding position in the field
- + gives a + if the number is positive, - if negative
- gives a - if the number is negative, space if positive

One level of parentheses may be used to indicate the number of occurrences of a format specification symbol. (E.g., X(6)BX(6) may be used instead of XXXXXXBXXXXXX, and -(7)9.99 instead of -----9.99 .)

For further info refer to the COBOL Reference Manual (ND-60.144).

In addition to the legal COBOL formats there is an extra feature: any alphanumeric string enclosed by apostrophes may appear at any place inside the format specification. This string will be displayed by FOCUS-RTS when editing of the field is completed. If an apostrophe is to be a part of the string, it must be followed by an additional apostrophe.

For numeric E-format fields the format is

Sk.mE+99

where the symbol 'S' represents an optional sign ('+' or '-'), 'k' and 'm' represents the significand, each may consist of zero or more occurrences of the symbol '9'. 'E' is an insertion character delimiting the exponent part. The exponent consist of a sign + two occurrences of the symbol '9'.

The format for Text fields is

TEXT(n)

where n is the number of characters on one text line. To specify additional lines, use the copying field function (see p. 43).

Format specification examples:

<u>Format</u>	<u>Data</u>	<u>Edited result</u>
999+	34	034+
-999	34	034
-999	-34	-034
ZZZ9	34	34
9.99E+99	34	3.40E+01
Z(6).999+	34.34	34.340+
ZZZ,ZZZ	34	34
ZZZ,ZZZ	0	
,+	-3434	**3,434-
-----9	-34	-34
XX/XX	abcd	ab/cd
909B9,9/	3434	304 3,4/
'\$'***,***.99	3434.34	\$**3,434.34
'Month: '29' Day: '99	224	Month: 2 Day: 24
X(4)' '*' 'X(5)	LeifKruise	Leif '*' Kruise
TEXT(60)	Norsk Data	Norsk Data

NOTE: in these examples the number group separator (when used) is set to , (comma). The decimal point is set to . (dot).

4.3.2 DEFINING OTHER FIELD ATTRIBUTES

The field attributes that can be defined for the different field types are listed below:

Field Attribute	Num	Num-E	String	Text
Storage type	*	*		
Legal characters			*	*
Justification			*	
Blank when zero	*	*	*	
Help form	*	*	*	*
Occurrence range	*	*	*	*
Initial value	*	*	*	
User control before edit	*	*	*	*
Default value	*	*	*	
System routine	*	*	*	
Legal values	*	*	*	
Illegal values	*	*	*	
User control after edit	*	*	*	*

These attributes are more thoroughly explained on p. 37.

4.3.3 STORAGE TYPES

The storage types in FOCUS and what they corresponds to in COBOL and FORTRAN:

FOCUS	COBOL	FORTAN
INT2	COMP (ND-100, PIC def. omitted) COMP PIC S9(n), n<=4	INTEGER (ND-100) INTEGER*2
INT4	COMP (ND-500, PIC def. omitted) COMP PIC S9(n), n>=5 and n<=10	INTEGER (ND-500) INTEGER*4 DOUBLE INTEGER
REAL4	COMP-2 (ND-100/32, PIC def. omitted) COMP-2 PIC S9(n).9(m), m+n<=6 (ND-500)	REAL (ND-100/32) REAL*4 (ND-100/32) REAL*4 (ND-500)
REAL6	COMP-2 (ND-100/48)	REAL (ND-100/48) REAL*6 (ND-100/48)
REAL8	COMP-2 PIC S9(n).9(m), m+n>=7 (ND-500)	REAL*8
BCD	PACKED-DECIMAL COMP-3	----
ACD	DISPLAY	----

Default storage type is ACD for numeric fields, REAL8 for E-format fields.

For E-format fields, REAL4, REAL6 and REAL8 are the only legal storage types.

Internal Representation

For the description of the internal representation of INTx and REALx, see the ND FORTRAN Reference Manual (ND-60.145). For the description of BCD, see the COBOL Reference Manual (ND-60.144).

When using storage code ACD (ASCII Coded Decimal), the digits are stored right-justified as ASCII characters. The fill character (for unused digit positions) is ASCII 0 (60B). If a sign is specified in the format, the last (rightmost) character position will contain a "+" or a "-".

Range and precision

Storage type	Range
INT2	-32,768 to 32,767
INT4	-2,147,483,648 to 2,147,483,647
REALx	-1.0 E+76 to 1.0 E+76 For the exponent part: -67 to +76
BCD	18 digits
ACD	80 digits

The precision is for

REAL4 - 6 digits
REAL6 - 9 digits
REAL8 - 15 digits

4.4 MODIFYING FIELDS

To modify a field definition, the same questionnaire is used as when you define a field. Position the cursor on the field, press the FIELD key, and then modify the definition.

4.5 MOVING, COPYING AND DELETING FIELDS

To move, copy or delete a field, you must first mark it. Position the cursor anywhere inside the field, and press the MARK key once. Although only this one position in the field is shown in inverse video, the whole field is now marked. As a general rule, if any part of a field is shown as marked, the whole field is marked. When you move the cursor to a different line, the marking will not show on the screen, but the field remains marked.

Moving, copying or deleting the marked field is done as in PED/NOTIS-WP: to copy or move the field, first move the cursor to the new start position for the field, and use the COPY key or the MOVE key. To delete the field, use the DELETE key inside the field.

4.5.1 OCCURRENCES

Copying a field creates occurrences of the old field. The occurrences are distinguished by occurrence numbers. The numbering of the occurrences is determined not by the sequence in which they are created, but by their position in the form itself, and by the occurrence range attribute. This means that the occurrence number may change if the occurrence of a field is moved.

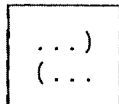
When occurrences are specified, the syntax is:
<field name>.<occurrence number>.

Example: XFIELD.3

See also the section on groups, p. 27.

4.6 DEFINING FORM EXTENT

If you do not specify the extent of a form, the form on the screen will occupy the smallest rectangle which can contain both the leading text and the fields. Before a form is written to the screen, this area will be cleared. To speed up this clearing, the form extent should be defined to contain the rightmost column on the screen, because then the terminal feature 'clear to end of line' will be used. Otherwise, the area must be cleared by filling it with spaces, which is much more time consuming. If you want the form to fill more of the screen, you can define the form extent by first marking one corner with the MARK key and then using the key



in the diagonally opposite corner.

CHAPTER 5

THE RUNTIME LIBRARY

5 THE RUNTIME LIBRARY

5.1 INTRODUCTION

This part of the manual describes the call interface to the FOCUS Runtime System, FOCUS-RTS.

For a user application, FOCUS-RTS offers a call interface to format a screen terminal and perform input/output from/to the formatted screen. Formatting of the screen is done by FOCUS, using predefined forms. These forms are defined using FOCUS-DEF, which is described in the first part of this manual. FOCUS fetches form descriptions as requested from the user application, and uses them to format the screen. Forms can be loaded with the program or read from a form file.

A detailed description of the routines in FOCUS-RTS is given in the following sections. The routines are grouped into the following categories:

- Initiation and definition calls.
- Read / modify field(s)
- Display field(s)
- Write document to file
- Message calls
- Define terminal operator interface
- Information / status calls
- Multi forms handling
- Other form related calls
- Form independent calls

A short description of the routines introduces each section.

All the routines have an output parameter called "status". This output parameter is assigned an error number, if an error is detected during the execution of the routine. If no error is detected, the output parameter is given the value zero. Format: I. (See section 5.3).

The names of fields, forms, etc., can be given in uppercase or lowercase when used as parameters.

NOTE: on the ND-500, because of optimization, leading texts, fields etc. written to the terminal using FOCUS calls, will be buffered. The text will be displayed on the terminal:

- on request for input using a FOCUS call
- when the buffer is full (the size is 2000 bytes)
- when FOCUS is terminated
- if the routine FCEBUF is called (see p. 106)

5.2 TERMS USED IN THIS CHAPTER

The reader should be familiar with the following terms:

Leading text, is an informative or explanatory text defined when a form is created. It is a permanent part of the form and cannot be modified by an application program at runtime.

Field, is a number of positions reserved for data input/output. A form can have several types of fields. The characteristics of a field are defined when it is created.

Form, is the combination of leading texts and fields, defined by FOCUS-DEF. A form has an 8 character (maximum) name which must be unique within any set of forms used simultaneously in one application.

Data element, is the internal representation of the characters displayed or typed in a field. For numeric fields, the internal representation is integer2, integer4, real4, real6, real8, BCD (binary coded decimal) or ACD (ASCII coded decimal), depending on the storage type. String fields are represented as ASCII bytes.

Data record field set: is a collection of all or a subset of the fields in a form. When a form is declared by "Declare form name" the data record field set will contain all fields in the form. "Declare record" can redefine the data record field set to contain a subset of the fields in the form.

Data record, is a collection of data elements corresponding to the fields in the data record field set.

Edit set, is the fields currently being edited.

Occurrence number. When a new field is defined it must be given a name by the terminal operator. Copies of this field may be created using the COPY function. This fields will get the same name as the "original" field, but they can be distinguished by occurrence numbers. The numbering is determined by their sequence in the form and the "occurrence range" parameter, not by the sequence in which they were created. The occurrence number range will be from one and upwards.

Field name: is a byte string of maximum 8 characters which identifies a field in a form and, if the field has been copied, all occurrences of the field. Legal characters in field names are letters, digits and underscore. The first character must be a letter.

Occurrence part. An occurrence part can be given in addition to a field name to identify one or several occurrences of a field. The occurrence part starts with a period (.) followed by an occurrence number or an occurrence range. An occurrence range consists of a first and last occurrence number separated with colon (:).

Examples:

FIELD1.1 FIELD2.3 FIELD3.1:5

Field group, is defined during the definition of a form. A field group can be regarded as a sequence of field names given with or without occurrence parts.

Field group name, is the name of a field group. The syntax of a field group name is the same as for field name.

Field list, is a byte string used as parameter to specify a set of fields. The string contains field names and/or field group names separated by one or several spaces. Field names can be given with or without occurrence parts.

Two parameters, "first occurrence number" and "last occurrence number", will always be given in addition to a "field list" parameter. If field names without occurrence parts are used in "field list", occurrence numbers will be taken from these parameters. All corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be included in the field set.

The following characters have a special meaning when given in the first position:

"*" - All fields. "All fields" refers to all fields in the data record field set, except when used from FCDECRC where "all fields" refers to all fields in the form.

"@" - The same set of fields that was used in the last call to FCDECRC, FCESUB, FCWSUB, FCGSUB, FCPSUB, FCCLSUB, FCSETAT, FCSETMR, FCCLMR or FCWDOTS. (Calls with a "field list" parameter.)

If any of the above calls are used from the user control routine (FCUCONT) or the user function routine (FCUFUNC), the "@" set of fields will be reset upon return.

"-" - All fields except fields specified in "field list". "All fields" refers to all fields in the data record field set, except when used from FCDECRC where "all fields" refers to all fields in the form.

The string must always be terminated by an apostrophe (').

In some of the FOCUS subroutines using the field list parameter the order of the fields will be of importance. The order is as follows:

- First the occurrence number is set to "first occurrence number" and the field list is traversed from the beginning. If a field name is given with an occurrence part, the occurrence number or occurrence range given with the field name is used. If no occurrence part is given, the first occurrence number is used.
- Then the occurrence number is increased by one, and the field list is traversed from the beginning again, but this time only fields indicated with a field name without occurrence part are selected.
- This is continued until the occurrence number has reached "last occurrence number".

The field order is shown in some examples below. The fields F2, F3 and F4 are copied using the COPY function in FOCUS-DEF.

F1

....

F2

F3

F4

.....

.....

.....

.....

.....

.....

.....

.....

.....

If the field list contains "F1.1 F2 F3 F4'", "first occurrence number" is 1, and "last occurrence number" is 3, the field order will be: F1.1, F2.1, F3.1, F4.1, F2.2, F3.2, F4.2, and so on.

If the field list contains "F1.1 F2.1:3 F3.1:3 F4.1:3'", the field order will be: F1.1, F2.1, F2.2, F2.3, F3.1, F3.2, and so on.

Key value, is the internal code generated for a key on the terminal keyboard.

Navigation, is moving from one field to another during editing.

5.3 USING FOCUS FROM COBOL, FORTRAN AND PLANC

In FOCUS-RTS four types of parameters are used. They are denoted by $X(n)$, $X(a,b)$, I and $I(n)$. The tables below show the corresponding types in COBOL, PLANC and FORTRAN:

FOCUS	COBOL	PLANC
$X(n)$	PIC X OCCURS n TIMES	BYTES : $X(0:n-1)$
$X(a,b)$	PIC X(a) OCCURS b TIMES	BYTES ARRAY : $X(b,a)$
I	COMP	INTEGER : I
$I(n)$	COMP OCCURS n TIMES	INTEGER ARRAY : $I(0:n-1)$

FOCUS	FORTRAN
$X(n)$	1. INTEGER*2 $X(i)$ where $i = (n+1)/2$ 2. CHARACTER STR*n EQUIVALENCE (STR,X(1)) 3. Hollerith constant, e.g. "FIELD3'"
$X(a,b)$	INTEGER*2 $X(i)$ where $i=((a*b)+1)/2$
I	INTEGER
$I(n)$	1. INTEGER $I(n)$ 2. INTEGER I DIMENSION I(n)

In COBOL, all parameters must be word-aligned on the ND-100.

When using FOCUS from PLANC, the routines must be declared as type STANDARD. Arrays used as parameters should be declared with lower index bound 0. Sub-arrays should not be used as parameters.

5.4 INITIATION AND DEFINITION CALLS

This chapter describes the following routines:

- FCINITE is used to initiate and terminate FOCUS.
- FCDECFE is used to tell FOCUS from which form file form descriptions are to be taken.
- FCDECFN defines which form is to be used in the subsequent FOCUS calls.
- FCDECRC redefines the format of a data record.
- FCSETAT is used to define display attributes (inverse video, blink etc.) on fields.
- FCSETMR/FCCLMR can be used to set/reset "Must read".
- FCNXFLD defines the field where the editing should start in the next edit call. The cursor position in the field may also be specified.
- FCESFNC can be used to execute a break function.
- FCPOSFO redefines the start position of the current form.

5.4.1 INITIATE / TERMINATE FOCUS FCINITE

This routine must be the first and the last call a user program makes to the screen handling system. When FCINITE is used to terminate FOCUS, the device number must be set to zero.

FCINITE(initiation array, form buffer, status)

Input parameters: initiation array, consists of 10 elements.

The array has the following layout:

Element 1: Length in number of bytes of the form buffer (second parameter).

Element 2: Logical device number (N) for the terminal.

N > 1 The device is to be reserved by FOCUS.

N = 1 The background terminal is used.

N = 0 FOCUS is terminated. The terminal will be released if reserved by FOCUS.

N < 0 The absolute value of the device number is used, but the device is not reserved.

Element 3: Not used, must be set to zero.

Element 4: Not used, must be set to zero.

Element 5: Word length (L). The data elements in a data record are always word aligned. This parameter tells FOCUS which word length to use. Legal values are:

L = 1: Word length is one byte, i.e., the data elements are packed.

L = 2: Word length is two bytes (16 bits), i.e., the last byte in the last word used by a data element is unused if the length of the data element is an odd number of bytes.

L = 4: Word length is four bytes (32 bits), i.e., up to three of the last bytes in the last word used by a data element may be unused.

Elements 6 - 10: Not used, must be set to zero.

Format: I(10).

form buffer, is used internally by FOCUS to store form descriptions read from a form file, and some status information connected to the current form.

The minimum size of the form buffer is the size needed to contain the biggest form to be used in the application. If FS is the size of the form description found using the command "LIST-FORMS" in FOCUS-DEF, N is the number of fields in the form and NCW is number of bytes in a machine word, then the buffer size needed for a form read from a form file is:

$(FS + 4*N*NCW + 70)$ bytes

If a form is loaded with the program or when a form is "saved", the form description will not occupy any space in the form buffer. The space needed for a "saved" form or for a loaded form when it is current is:

$(4*N*NCW + 70)$ bytes

The buffer size should, however, be as big as possible so that several forms can be kept in the buffer simultaneously.

Format: X(Length of form buffer).

FCINITE can also be used to reset the screen handling system to the state it had before the first routine (other than FCINITE) was called.

If a terminal using escape sequences is used, MON71 (disable escape) will be executed.

When FCINITE is called with the device number equal to zero, the form file will be closed, MON72 (enable escape) will be executed and the terminal's function switches (if any) will be set back to their default values. NOTE: on some terminal types, this will cause the screen to be cleared.

5.4.2 DECLARE FORM FILE FCDECF

This routine tells FOCUS from which form file form descriptions are to be taken.

FCDECF(form file name, status)

Input parameter: form file name, is the SINTRAN file name of the form file from which the forms are to be taken. The parameter must be left justified, and terminated by an apostrophe ('). Format: X(length of file name).

The routine opens the file, checks that the file is a form file, and stores the file number for later use. If another form file is already opened, it will be closed before the new one is opened.

When running on ND-500 the form file will be connected as a segment.

An empty string used as "form file name" will close an already opened form file.

5.4.3 DECLARE FORM NAME FCDECFN

This routine is used for choosing a form to be used in the subsequent FOCUS calls. A "saved" form will be re-activated. The form description must reside on the last declared form file, or be loaded with the program. The form will be written to the screen on the next call to FCEREC, FCESUB, FCEFLD, FCWREC, FCWSUB, FCWFLD, FCWLTX, FCWDOTS or FCCLFDS.

FCDECFN(form name, mode, status)

Input parameters: form name, is the name of the form to be used in subsequent FOCUS calls. The parameter must be left justified and terminated by an apostrophe. Format: X(length of form name).

mode, there are four possible values:

- 1: No clear before displaying the leading texts.
- 0: Clear the whole screen before displaying the form.
- 1: Clear only the part of the screen used by the form before displaying the form.
- 2: No clear and no display of leading texts.

Format: I.

The routine makes the given form ready for use.

After a call to FCDECFN, the data record field set will contain all the fields in the form, if the form was not a "saved" form.

5.4.4 DECLARE RECORD FCDECRC

This routine is used to define the data record field set as a subset of the fields in the form, i.e., the format of a data record will be redefined.

FCDECRC(field list, first occurrence number, last occurrence number, status)

Input parameters: field list, contains field names for all fields to be included in the data record field set. See p. 52 for the definition of the field list parameter.
Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be included. Format: I.

5.4.5 SET DISPLAY ATTRIBUTES FCSETAT

This routine is used to set display attributes (inverse video, blink etc.) on fields in the data record field set.

FCSETAT(attribute code, field list, first occurrence number,
last occurrence number, status)

Input parameters: attribute code, the bits 0 - 7 (bits are numbered from right to left) represent the various display attributes:

<u>Bit</u>	<u>Value</u>	<u>Display attribute</u>
none	0	Normal
0	1	High (increased) intensity
1	2	Low (decreased) intensity
2	4	Italics
3	8	Underlined
4	16	Blink (slowly)
5	32	Blink (rapidly)
6	64	Inverse video
7	128	Invisible

Attributes can be combined by adding the values corresponding to the display attributes wanted. (For example, attribute code $64 + 8 = 72$ will give the display attribute Inverse video + Underline).
Format: I.

field list, contains field names for the fields to be given a new attribute. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be used. Format: I.

The attributes will appear the next time the fields are displayed.

Display attributes on fields not specified in the "field list" will not be affected.

Note: for the time being, display attributes are only available on a limited number of terminals. (See appendix B for a description of the various terminals.) For other terminals, attributes are ignored.

5.4.6 SET MUST READ FCSETMR

This routine is used to define fields in the data record field set as "must read" fields. FOCUS will check that all "must read" fields in the edit set are given a value, or that carriage return is used to leave the field, before leaving an edit call (FCEREC, FCESUB or FCEFLD).

FCSETMR(field list, first occurrence number, last occurrence number, status)

Input parameters: field list, contains field names identifying the "must read" fields. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be used. Format: I.

Only the fields in the "field list" will be affected.

5.4.7 CLEAR MUST READ FCCLMR

This routine is used to reset "must read" fields.

FCCLMR(field list, first occurrence number, last occurrence number,
status)

Input parameters: field list, contains field names identifying the fields to be reset. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be used. Format: I.

Only the fields in the "field list" will be affected.

5.4.8 DEFINE NEXT FIELD TO BE EDITED FCNXFLD

This routine is used to define the field where the editing should start in the next FCEREC or FCESUB call. The cursor position in the field may also be specified. The routine may also be used from a user control or user function routine to specify the next field to be edited, and the cursor position in this field.

FCNXFLD(field name, occurrence number, position, status)

Input parameters: field name, is the name of the field where the editing is to start. The field name can be given with or without occurrence part and must end with an apostrophe ('). Format: X(length of field name).

occurrence number, indicates which occurrence of the field is desired if no occurrence part is given in the field name. Format: I.

position, is the position in the field where the cursor is to be positioned. Format: I.

When FCEREC or FCESUB are called, the editing will normally start in the first field in the edit set. This can be changed using the routine FCNXFLD. The change will only affect the first call to FCEREC or FCESUB and will have no other effect than defining the field in which the editing should start. The routines FCDECFN, FCEREC, FCESUB and FCEFLD will reset the edit start field to the first field in the edit set. See also the FCESFNC call.

NOTE: if the field name given does not exist in the edit set, an error status is returned from the edit call, not from the FCNXFLD call. This routine replaces the routine FCESFLD. A new parameter 'position' is added. The FCESFLD routine will still be available in the library, but will no longer be described in the manual. For the description of the FCESFLD call, see the old manual.

5.4.9 DEFINE EDIT START FUNCTION FCESFNC

When FCEREC or FCESUB is called, editing will start in the first field. This can be changed using this routine.

The edit start function will be performed from the current field. After an edit call, the current field is the field where the editing was terminated.

FCESFNC(break function number, line, column, status)

Input parameters: break function number, is the number of the break function to be performed, before the editing is started in the next edit call. Format: I.

line and column, specify a cursor position. If one of these parameters is given a value different from zero, the function will be performed from the cursor position instead of from the current field. Only some break functions can be performed from the cursor position. These functions are LEFT1 (1002), RIGHT1 (1003), UP1 (1004), DOWN1 (1005), UP (1010), DOWN (1011) and LEFT\$ (1012). Format: I.

The change will only affect the first call to FCEREC or FCESUB. The routines FCDECFN, FCEREC, FCESUB and FCEFLD will reset the edit start field to the first field in the edit set.

These routines may also be used from a user control routine or user function routine to define a break function to be performed before the editing continues.

A entitled description of the various break functions is given in the section Break Functions on page 77.

5.4.10 POSITION FORM FCPOSFO

This function may be used to redefine the start position of current form.

FCPOSFO(line, column, status)

Input parameters: Line and column. This is the new start position of the form. Format: I.

The call can also be used to bring an overlapped form to the top if the multi forms handling calls are used.

5.5 READ / MODIFY FIELD(S)

This section describes how fields in a form can be edited. Three routines are available:

- FCEREC is used to edit all fields in the data record field set.
- FCESUB is used to edit one or several of the fields in the data record field set. The parameter "field list" specifies which fields are to be edited. NOTE: the routine will operate on a data record containing the data elements for all fields in the data record field set.
- FCEFLD is used to edit a single field on the screen. As opposed to the routines above, this routine will not operate on a data record, only on the data element for this field. The field need not be in the data record field set.

The following functions will be performed:

- For a new form, the screen is cleared according to the "mode" parameter in the last call to FCDECFN.
- For a new form, the leading texts in the form are displayed if the "mode" parameter in the last call to FCDECFN is unequal to 2.
- Depending on the parameter "edit mode" :

edit mode = 0 MODIFY

- The data elements are displayed.
- Editing can start.

edit mode = 1 NORMAL READ

- The data elements are cleared. (Numeric fields are given the value zero; text fields are filled with spaces.)
- Dots are displayed in the fields.
- Editing can start.

edit mode = 2 CONTINUE READ/MODIFY

- Editing can start.

It is assumed that the fields are displayed already.

edit mode = 3 NORMAL READ WITHOUT DISPLAYING DOTS

- The data elements are cleared. (Numeric fields are given the value zero; text fields are filled with spaces.)
- Editing can start.

It is assumed that dots are displayed in the fields already.

edit mode = 4 READ PASSWORD

- The data elements are cleared. (Numeric fields are given the value zero; text fields are filled with spaces.)
- Dots are displayed in the fields.
- Editing with no echo can start.

edit mode = 5 CONTROL READ

- Dots are displayed in the fields.
- Controlled editing can start.

When a field is read it is compared with the value in the corresponding data element. If it does not match, a "beep" is given and the old value will be displayed on the message line. The operator can now:

- 1) Terminate the field once more.
- 2) Continue to edit the field.
- 3) Press the CANCEL key to get the old value.

The routines FCDESB and FCEDSTA can be used to get edit status information. See sections 5.10.1 and 5.10.2.

Editing is described in chapter 6.

5.5.1 EDIT RECORD FCEREC

This routine is used to edit all fields in the data record field set.

FCEREC(data record, edit mode, status)

Input parameters: data record, contains the data elements corresponding to the fields in the data record field set. Format: X(length of data record).

edit mode, specifies the edit mode. Format: I.

Output parameter: data record, returns the edited data record.

The editing will start in the first field in the data record field set. This can be changed using the routines FCNXFLD or FCESFNC.

5.5.2 EDIT SUBRECORD FCESUB

This routine is used to edit a subset of the fields in the data record field set.

FCESUB(field list, first occurrence number, last occurrence number,
data record, edit mode, status)

Input parameters: field list, contains field names for all fields to be edited. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be used. Format: I.

data record, contains the data elements corresponding to the fields in the data record field set. Format: X(length of data record).

edit mode, specifies the edit mode. Format: I.

Output parameter: data record, returns the edited data record.

The editing will start in the first field in the field list. This can be changed using the routines FCNXFLD or FCESFNC.

If the subset is set to contain all fields in the data record field set (field list="*"), this call is identical to the FCEREC call.

NOTE that the redisplay function will redisplay all fields in the data record field set.

5.5.3 EDIT ONE FIELD FCEFLD

This routine is used to edit a single field from the last declared form.

FCEFLD(field name, occurrence number, data element, data element length, edit mode, status)

Input parameters: field name, contains a field name identifying the field to be edited. The field name can be given with or without an occurrence part and must be terminated by an apostrophe ('). Format: X(length of field name).

occurrence number, are used to indicate which occurrence of the field is desired if no occurrence part is given in field name. Format: I.

data element, contains the data element corresponding to the field. Format: X(length of data element).

edit mode, specifies the edit mode. Format: I.

Output parameters: data element, returns the edited data element.

data element length, returns the length of the edited data element. If the edited field is a numeric field, maximum length will always be returned. Format: I.

FCEFLD can be used even if the given field is not in the data record field set.

NOTE: when you use the redisplay function (FUNC @), or when the form is redisplayed after being overwritten by a help-form, only the field specified in this call will be redisplayed. If this causes problems, use the FCESUB call instead.

5.6 DISPLAY FIELD(S)

This section describes how fields in a form can be written to the screen. Three routines are available:

- FCWREC is used to display on the screen all fields in the data record field set.
- FCWSUB is used to display on the screen one or several of the fields in the data record field set. The parameter "field list" specifies which fields are to be displayed. NOTE: the routine will operate on a data record containing the data-elements for all fields in the data record field set.
- FCWFLD is used to display a single field on the screen. As opposed to the routines above this routine will not operate on a data record, only on the data element for this field. The field need not be in the data record field set.

The following actions will be performed:

- For a new form the screen is cleared according to the "mode" parameter in the last call to FCDECFN.
- For a new form the leading texts in the form are displayed if the "mode" parameter in the last call to FCDECFN is unequal to 2.
- The field(s) are displayed.

The fields can be displayed with display attributes (inverse video, blink etc.). See section 5.4.5 for how to set display attributes.

5.6.1 WRITE RECORD FCWREC

This routine is used to display all fields in the data record field set on the screen.

FCWREC(data record, status)

Input parameter: data record, contains the data elements corresponding to the fields in the data record field set. Format: X(length of data record).

5.6.2 WRITE SUBRECORD FCWSUB

This routine is used to display a subset of the fields in the data record field set on the screen.

FCWSUB(field list, first occurrence number, last occurrence number,
data record, status)

Input parameters: field list, contains field names for all fields to be displayed. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be used. Format: I.

data record, contains the data elements corresponding to the fields in the data record field set. Format: X(length of data record).

If the subset is set to contain all fields in the data record (field list="*"), this call is identical to the FCWREC call.

5.6.3 WRITE ONE FIELD FCWFLD

This routine is used to display a single field of the last declared form.

FCWFLD(field name, occurrence number, data element, data element
length, status)

Input parameters: field name, contains a field name identifying the field to be displayed. The field name can be given with or without an occurrence part and must be terminated by an apostrophe ('). Format: X(length of field name).

occurrence number, is used to indicate which occurrence of the field is desired if no occurrence part is given in the field name. Format: I.

data element, contains the data element to be displayed. Format: X(length of data element).

data element length, is the length of the data element in number of bytes. If set to -1 or zero, maximum length is used. For numeric fields, maximum field length should always be used. Format: I.

5.7 WRITE DOCUMENT TO FILE

This chapter describes the calls needed to write documents to a file or a line printer.

- FCOPEN opens the document file and returns a file number to be used by FCPRDOC.
- FCPRDOC writes the document to the specified file.
- FCCLOSE closes the document file.

5.7.1 OPEN FILE FCOPEN

This routine is used to open a file to be used in a call to FCPRDOC.

FCOPEN(file name, access code, file number, status)

Input parameters: file name, SINTRAN file name of the file. The parameter must be left justified, and terminated by an apostrophe ('). Format: X(length of file name).

access code, contains the SINTRAN access code. The parameter must be left justified, and terminated by an apostrophe ('). Possible values are "W" (write) and "WA" (write append). Format: X(length of access code).

Output parameter: file number, is the SINTRAN file number for the opened file. Format: I.

5.7.2 PRINT DOCUMENT ON FILE FCPRDOC

This routine is used to write a form with leading texts and fields to a file.

FCPRDOC(data record, file number, status)

Input parameters: data record, contains the data elements corresponding to the fields in the data record field set. Format: X(length of data record)

file number, is the SINTRAN III file number of the output file. This file must previously have been opened for output by a FCOPEN call. Format: I.

A file containing characters from the alternative character set (graphic, Greek) should be written to a printer with the PRINT command in NOTIS-WP.

To be able to read a file containing characters from an alternative character set in NOTIS-WP, the file must start with the following four characters: "#ID," where "#" represents the octal value 340b.

5.7.3 CLOSE FILE FCCLOSE

This routine is used to close a file.

FCCLOSE(file number, status)

Input parameter: file number, is the SINTRAN file number for the opened file, output parameter from FCOPEN. Format: I.

5.8 MESSAGE CALLS

This chapter describes routines used to exchange messages between the application program and the terminal operator.

- FCZMSGE is used to display a message on the message line.
- FCGMSGE is used to write a leading text and then read a message (read a command, parameter etc.), from the terminal operator.

The second last line on the screen is used as the message line.

This message line is also used by FOCUS to give messages to the terminal operator during the editing (Illegal value, Illegal date, etc.).

The message line will be cleared when leaving a field during the editing. It will also be cleared when a new message is written.

If the application program wants to control the clearing of messages, the FCWTEXT and FCRTXT routines can be used.

5.8.1 SEND MESSAGE FCZMSGE

This routine is used to give a message to the terminal operator.

FCZMSGE(message, status)

Input parameter: message, is the message to be displayed. It must be left justified, maximum 80 characters, and terminated by an apostrophe. Format: X(length of message).

The Send Message routine clears the message line, and then displays the message from the beginning of this line.

5.8.2 GET MESSAGE FCGMSGE

This routine is used to read a message from the terminal operator after having displayed a leading text.

FCGMSGE(leading text, message, status)

Input parameter: leading text, a leading text to be displayed before the message. It must be left justified, maximum 80 characters and terminated by an apostrophe.
Format: X(length of leading text).

Output parameter: message, returns a message typed on the screen after the leading text. The maximum length of the message is 80 minus the number of characters in the leading text, i.e., 80 characters maximum.
Format: X(80).

Get message performs the following functions:

- The message line is cleared and the leading text is displayed from the beginning of this line.
- The cursor is placed in the first free position after the leading text, allowing the terminal operator to enter a message, which can consist of any alphanumeric characters. The terminal operator entering the message may use the same editing facilities as when editing a field in a form. See chapter 6.
- The reading is terminated by pressing carriage return.

5.9 DEFINE TERMINAL OPERATOR INTERFACE

5.9.1 BREAK AND BREAK FUNCTIONS

A break is in this section defined as an interrupt of the editing of a field. This interrupt will cause a function (break function) to be performed. This can either be a return to the user program, or it can cause a FOCUS or user defined function to be performed before the editing continues.

FOCUS has a set of break functions connected to certain break conditions. This set defines the navigation strategy, the conditions for a return to the user program, and the terminal keys used to perform the help, the redisplay, the cancel and the copy functions.

It may be redefined by the programmer by connecting functions, specified by function numbers, to break conditions. The following routines can be used:

- FCCHRB is used to define function keys, i.e., connect break functions to terminal keys.
- FCFLDB is used to define the break functions to be performed when "leaving field" situations occur; for example when cursor right is entered in the last position of a field.
- FCFRMB is used to define the break functions to be performed when "leaving form" situations occur; for example when next field is requested in the last field in the edit set.

5.9.2 BREAK FUNCTIONS

The available functions together with their function numbers will be described in this section. New functions may be added in later versions of FOCUS.

5.9.2.1 RETURN TO USER PROGRAM FUNCTIONS

(nn) ERROR TERMINATION: This function will lead to an immediate return to the user program. No further checking will be done, and the data element for the current field will not be updated. (nn) is a number in the range -1 to -299 and will be found as the termination code after the break.

(nn) NORMAL TERMINATION: This function gives a controlled return to the user program. Before returning, value checks will be performed, the user control routine called and "must read" fields checked. (nn) is a number in the range 0 to 299 and will be found as the termination code after the break.

5.9.2.2 FUNCTIONS FOR NAVIGATION IN A FORM

The following functions will always operate on the fields in the edit set. NEXT and PREVIOUS follow the order of the fields in the edit set.

Some of these functions may cause a "leaving form" situation. Such a situation will cause a new function to be performed, namely the function specified for that situation. See section 5.9.5.

1000 NEXT: The cursor is moved to the first position in the next field. If the function NEXT is given in the last field, a situation called OUT NEXT is entered. Further action is then decided by the function defined for that situation.

1001 PREVIOUS: The cursor is moved to the first position in the previous field. If PREVIOUS is given in the first field, a situation called OUT PREVIOUS is entered. Further action is then decided by the function defined for that situation.

- 1002 LEFT1: The cursor is moved to the first position in the nearest field on the left side on the same line. If the function LEFT1 is given in the leftmost field of a line, a situation called OUT LEFT is entered. Further action is then decided by the function defined for that situation.
- 1003 RIGHT1: The cursor is moved to the first position in the nearest field on the right side on the same line. If the function RIGHT1 is given in the rightmost field of a line, a situation called OUT RIGHT is entered. Further action is then decided by the function defined for that situation.
- 1004 UP1: The cursor is moved to the first position in the field above. If there is no field immediately above it, the cursor is moved to the nearest field on the right side on the line above. If there are no fields on the right side, the cursor is moved to the nearest field on the left side. If there are no fields at all on this line, the line above this will be tried. If there are no fields on lines above, a situation called OUT UP is entered. Further action is then decided by the function defined for that situation. If inside a text field, the function UP (1010) will be performed.
- 1005 DOWN1: The cursor is moved to the first position in the field below. If there is no field immediately below it, the cursor is moved to the nearest field on the right side on the line below. If there are no fields on the right side, the cursor is moved to the nearest field on the left side. If there are no fields at all on this line, the line after this will be tried. If there are no fields on lines below, a situation called OUT DOWN is entered. Further action is then decided by the function defined for that situation. If inside a text field, the function DOWN (1011) will be performed.
- 1006 FIRST: The cursor is moved to the first position in the first field.

- 1007 LAST: The cursor is moved to the first position in the last field.
- 1008 FIRST THIS LINE: The cursor is moved to the first position in the leftmost field on the current line.
- 1009 LAST THIS LINE1: The cursor is moved to the first position in the rightmost field on the current line.
- 1010 UP: If there is a field immediately above, the cursor is moved to the position immediately above, otherwise the function UP1 is performed.
- 1011 DOWN: If there is a field immediately below, the cursor is moved to the position immediately below, otherwise the function DOWN1 is performed.
- 1012 LEFT\$: This function does the same as LEFT1, except that the cursor is moved to the last position of the field.
- 1013 LAST THIS LINE\$: This function does the same as LAST THIS LINE1, except that the cursor is moved to the last position of the field.

5.9.2.3 OTHER FUNCTIONS

- 2000 NO ACTION: No action is performed.
- 2001 BELL: A "beep" is given on the terminal.
- 2002 DISPLAY HELP: The help picture for the current field is displayed.

- 2003 REDISPLAY: The screen is cleared and the current form is redisplayed. "Saved" forms are redisplayed according to the redisplay flag in the FCSAVFO call.
- 2004 CANCEL: The contents of the field when the editing started are restored.
- 2005 COPY: The previous occurrence of the current field is copied to the field. The next field is entered.
- 2006 ACCEPT VALUE: This function can be used to force a value into a field without performing any legal value checks. If the function immediately following this function terminates the editing of the field (e.g. move to next field, user break etc.), no checks will be performed before leaving the field.
- NOTE that the user control routine will still be called. A flag in the Edit Info array will indicate if the accept value function is performed and can be checked upon.
- 2007 INSERT LINE: All lines (i.e. field occurrences) from and including this line will be moved one line downwards and a new blank line will be inserted. The function is only performed if the field is a text field.
- 2008 WORD-WRAP: The last word on this line (i.e. field occurrence) will be moved to the next line. The function is only performed if the field is a text field.
- 2009 DELETE LINE: If the field is a text field, the lines (i.e. field occurrences) following this line will be moved one line upwards. The last line will be cleared.

2010 REDISPLAY, NO CLEAR: The current form is redisplayed. "Saved" forms are redisplayed according to the redisplay flag in the FCSAVFO call. The screen is not cleared.

5.9.2.4 USER DEFINED FUNCTIONS

A function number in the range 9000 to 9999 will cause a subroutine with the following format to be called:

FCUFUNC(function number, field name, occurrence number,
edit buffer, length of buffer, number of
significant characters, cursor position,
changed flag, display flag, status)

Input parameters: function number, contains the function number.
Format: I.

field name, is the name of the field where the interrupt took place. Format: X(8).

occurrence number, is the occurrence number of the field where the interrupt took place. Format: I.

edit buffer, contains the screen image of the current field. Format: X(length of field).

length of buffer, specifies the length of the edit buffer. Format: I.

number of significant characters, specifies the number of significant characters in the edit buffer. Format: I.

cursor position, specifies the position of the cursor within the field. Format: I.

changed flag, has two possible values:

0: The field has not been changed after it was entered.

1: The field has been changed after it was entered.

Format: I.

Output parameters: function number, if this parameter is given a new value it will cause a new function to be performed.

edit buffer, new screen image.

number of significant characters, number of significant characters in the new screen image.

cursor position, new cursor position.

changed flag, must be set to 1 if the contents of the edit buffer are changed in the user routine. If there is no change, the flag should not be updated.

display flag, must be given the value 1 if FOCUS is to redisplay the field. Format: I.

User defined functions must be written in this format and loaded before FOCUS library is loaded.

Example of use: add new edit functions, for example, convert to uppercase/ lowercase. Connect runtime constants to terminal keys.

The following FOCUS routines can not be called from FCUFUNC:

FCCLFO	FCDECFF	FCDECFN	FCDECRC	FCDESB	FCEFLD
FCEREC	FCESUB	FCINITE	FCPOSFO	FCSAVFO	

5.9.3 DEFINE FUNCTION KEYS FCCHRRBR

Using this routine the programmer may connect functions to terminal keys. If keys used by FOCUS for editing are defined as function keys, the FOCUS edit function will be suppressed.

FCCHRRBR(number of function keys, key values, functions numbers,
status)

Input parameters: number of function keys, specifies the number of function keys to be defined. Format: I.

key values, consists of one element for each function key, containing the key value. Format: I (number of function keys).

function numbers, consists of one element for each function key, containing the number of the function to be connected. Format: I(number of function keys).

The predefined function keys and corresponding functions are:

"CR" ↵	(15B)	NEXT	(1000)
==>	(316B)	NEXT	(1000)
<==	(302B)	PREVIOUS	(1001)
FUNC "HOME"	(235B)	NORMAL TERMINATION	(1)
FUNC <W>	(227B)	ERROR TERMINATION	(-1)
HELP	(277B)	DISPLAY HELP	(2002)
FUNC @	(300B)	REDISPLAY	(2003)
CANCEL	(330B)	CANCEL	(2004)
<W>	(27B)	CANCEL	(2004)
COPY	(303B)	COPY	(2005)
<L>	(14B)	INSERT LINE	(2007)
F1	(204B)	DELETE LINE	(2009)
F2	(214B)	INSERT LINE	(2007)

When using this call, all the function keys which will be used and the corresponding functions must be specified, since information about previously defined function keys are lost.

The octal values of the keys and the function numbers are specified in parentheses. The command GET-KEY-VALUE in FOCUS-DEF can be used to get the values of the terminal keys.

5.9.4 DEFINE LEAVING FIELD FUNCTIONS FCFLDBR

Using this routine the programmer may define the function to be performed in the following situations:

- Cursor left in the first position of a field.
- Cursor right in the last position of a field.
- Cursor up in a field.
- Cursor down in a field.
- A character is entered in the last position of a field.
- Word-wrap required.
- Delete line required (CTRL-D CTRL-D).

FCFLDBR(number of leaving field situations, functions, status)

Input parameters: number of leaving field situations, specifies the number of leaving field situations to be defined. If this parameter is given a value less than 7, the functions to be performed for rest of the break situations remain unchanged. Format: I.

functions, contains function numbers corresponding to the situations mentioned above, given in the same order. Format: I(number of leaving field situations).

The predefined functions are:

LEFT1	(1002)
RIGHT1	(1003)
UP1	(1004)
DOWN1	(1005)
NO ACTION	(2000)
WORD-WRAP	(2008)
DELETE LINE	(2009)

The function numbers are specified in parentheses.

NOTE: if the function to be performed in the situation 'a character is entered in the last position of a field' is anything other than NO ACTION (2000), the break situation 'word-wrap required' will never occur at the end of the line.

5.9.5 DEFINE LEAVING FORM FUNCTIONS FCFRMBR

Using this routine the programmer may define the functions to be performed in the following situations:

```
OUT NEXT
OUT PREVIOUS
OUT RIGHT
OUT LEFT
OUT UP
OUT DOWN
```

FCFRMBR(number of leaving form situations, functions, status)

Input parameters: number of leaving form situations, specifies the number of leaving form situations to be defined. If this parameter is given a value less than 6, the functions to be performed for the rest of the break situations remain unchanged. Format: I.

functions, contains function numbers corresponding to the situations mentioned above, given in the same order. Format: I(number of leaving form situations).

The predefined functions are:

```
NORMAL TERMINATION  (  0)
NORMAL TERMINATION  (  2)
BELL                 (2001)
BELL                 (2001)
BELL                 (2001)
BELL                 (2001)
```

The function numbers are specified in parentheses.

5.10 INFORMATION / STATUS CALLS

This section describes the following routines:

- FCDESB is used to define a buffer for FOCUS where edit status information is to be put.
- FCEDSTA can be used to find out which fields were changed in the last edit call, and how the editing was terminated.
- FCSCRIN is used to get information about the screen terminal.
- FCFRMIN is used to get information about the current form.
- FCFLDIN is used to get information about a field.
- FCFLDNA can be used to get the names of the fields in a form.
- FCFLDST is used to get status information about a field.

5.10.1 DEFINE EDIT STATUS BUFFER FCDESB

This routine may be used to define a buffer for FOCUS where edit status information is to be put. This information will be transferred to the user program after each edit call (FCEREC, FCESUB or FCEFLD).

FCDESB(edit status buffer, number of elements, status)

Input parameters: edit status buffer, contains a record with the following format:

Element 1: Termination code which can be used to find out how the operator terminated the editing.

The default termination codes are:

-1: Editing terminated with FUNC <W>.

0: Editing terminated with \leftarrow or \Rightarrow in the last field.

1: Editing terminated with FUNC "home".

2: Editing terminated with \Leftarrow in the first field.

Format: I.

Element 2: Termination character. Format: I.

Element 3: Number of fields that were given a new value in the edit call. Format: I.

Element 4: Number of fields in the edit set. Format: I.

Element 5: Name of the field where the interrupt took place. Format: X(8).

Element 6: Occurrence number of the field where the interrupt took place. Format: I.

Element 7: Cursor position in the field. Format: I.

number of elements, specifies the number of elements to be transferred. If all information is of interest, the number of elements must be set to 7. If only the termination code is of interest, it should be set to 1. Format: I.

NOTE: the elements in the edit status buffer should not be accessed from a user control routine, because the buffer is updated when the edit call is terminated.

5.10.2 GET EDIT STATUS FCEDSTA

This routine is used to find out which fields were given a new value in the last edit call (FCEREC, FCESUB or FCEFLD), and how the editing was terminated.

FCEDSTA(number of fields edited, record status array, termination code, status)

Output parameters: number of fields edited, contains the number of fields that were given a new value in the last edit call. Format: I.

record status array, contains one element for each field in the data record field set. If an element has the value 0 (zero), the corresponding field has not been changed. If it has the value 1, the corresponding field has been changed. If the last edit call was FCEFLD, the array will have only one element. Format: I(number of fields in the data record field set).

termination code, is used to find out how the terminal operator terminated the editing.

The default termination codes are:

-1: Editing terminated with FUNC <W>.

0: Editing terminated with ↵ or ==> in the last field.

1: Editing terminated with FUNC "home"

2: Editing terminated with <== in the first field.

Format: I.

5.10.3 GET SCREEN INFORMATION FCSCRIN

This routine may be used to get information about the screen terminal.

FCSCRIN(screen info array, screen info array length, status)

Input parameter: screen info array length, specifies the number of elements in the Screen Info array. In this version this number must be 4. Format: I.

Output parameter: screen info array, consists of 4 elements. The array has the following layout:

Element 1: Terminal type number

Element 2: Message line number

Element 3: Number of lines/page

Element 4: Number of characters/line

Format: I(4).

5.10.4 GET FORM INFORMATION FCFRMIN

This routine may be used to get information about the current form.

FCFRMIN(form info array, form info array length, status)

Input parameters: form info array length, specifies the number of elements in the Form Info Array. In this version this number must be 6. Format: I.

Output parameters: form info array with 6 elements. The array has the following layout:

Element 1: Start line of the form on the screen.
Element 2: Start column of the form on the screen.
Element 3: Form size, number of lines.
Element 4: Form size, number of columns.
Element 5: Number of fields in the data record
 field set.
Element 6: Length of a data record.
Format: I(6).

5.10.5 GET FIELD INFORMATION FCFLDIN

This routine may be used to get information about a field.

FCFLDIN(field name, occurrence number, field info array, field
 info array length, status)

Input parameters: field name, contains a field name identifying the
field from which the information is to be
retrieved. The field name can be given with or
without an occurrence part, and must be terminated
by an apostrophe ('). Format: X(length of field
name).

occurrence number, is used to indicate which
occurrence of the field is desired if no occurrence
part is given in field name. Format: I.

field info array length, specifies the number of
elements in the field info array. In this version
this number must be 5. Format: I.

Output parameter: field info array, consists of 5 elements. The array
has the following layout:

Element 1: Y (line) position on screen.

Element 2: X (column) position on screen (start position).

Element 3: Length of field on screen.

Element 4: Displacement to data element in data record (in number of bytes).

Element 5: Length of data element (in number of bytes).

Format: I(5).

5.10.6 GET FIELD NAME FCFLDNA

This routine can be used to get the names of the fields in a form.

If each of the fields in the data record field set is given a number from one and upwards, and these numbers follow the same order as the fields in the data record field set, the routine can be used to convert these field numbers into field names.

FCFLDNA(from field number, to field number, field names, occurrence numbers, status)

Input parameters: from field number and to field number. Format: I.

Output parameters: field names. Format: X(8, number of field names)
occurrence numbers. Format: I(number of field names)

5.10.7 GET FIELD STATUS FCFLDST

This routine may be used to get status information about a field.

FCFLDST(field name, occurrence number, field status array,
field status array length, status)

Input parameters: field name, contains a field name identifying the field for which status information is desired. The field name can be given with or without an occurrence part and must be terminated by an apostrophe ('). Format: X(length of field name).

occurrence number, indicates which occurrence of the field is desired if no occurrence part is given in field name. Format: I.

field status array length, specifies the number of elements in the field status array. In this version this number must be 6. Format: I.

Output parameter: field status array, consists of 6 elements. The array has the following layout:

Element 1: Attribute code (indicating inverse video, blink, etc.).

Element 2: Must-read flag which is set to 1 if the field is a "must read" field, else the flag is set to 0.

Element 3: Value flag which is set to 1 if the field contains a value, else the flag is set to 0.

Element 4: New-value flag which is set to 1 if a new value was given to the field in the last edit call (or in the current edit call if this routine is called from a user control routine), else the flag is set to 0.

Element 5: Edit-set flag which is given the value 1 if the field is in the edit set, otherwise 0 (zero).

Element 6: Data-record-field-set flag which is set to 1 if the field is in the data record field set, else the flag is set to 0.

Format: I(6).

5.11 MULTI FORMS HANDLING

The routines in this section will make it easier to handle more than one form on the screen at a time.

- FCSAVFO will save status information about the current form.
- FCCLFO will remove the current form from the screen and redisplay overlapping forms.

5.11.1 SAVE FORM FCSAVFO

This function is used to save status information about the current form. This should be used if you want to display or edit other forms and then return to this form without losing information about display attributes, dots in the fields, redefined start position of the form etc.

FCSAVFO(redisplay flag, data record, status)

Input parameters: redisplay flag, has three possible values:

- 0: The form will not be redisplayed if the redisplay function is given when working with other forms.
- 1: The form will be redisplayed if the redisplay function is given. The last used data record will be used when redisplaying the fields.
- 2: The form will be redisplayed if the redisplay function is given. The data record supplied with this call will be used when redisplaying the fields.

Format: I.

data record, is a buffer containing data elements corresponding to the fields in the data record field set. This parameter is only relevant if the redisplay flag is set to 2. Format: X(length of data record).

The form is "saved" until the next call to FCDECFN using this form name. A maximum of ten forms can be "saved" by FOCUS simultaneously.

If the redisplay key (FUNC @) is used, or when returning from a help form, all forms "saved" using this call will be redisplayed according to the redisplay flag.

If the saved forms overlap on the screen, the last form "saved" or current form if FCPOSFO is used, will be displayed on top of the others.

After a call to FCSAVFO there will be no current form.

5.11.2 CLEAR FORM FCCLFO

This function may be used to remove current form from the screen. Any overlapping forms will be redisplayed.

FCCLFO(status)

5.12 OTHER FORM RELATED CALLS

The following routines are described in this section:

- FCCLREC clears all data elements in the data record.
- FCCLSUB clears a subset of the data elements in the data record.
- FCWDOTS displays dots in one or several of the fields in the data record field set.
- FCCLFDS displays dots in the fields.
- FCGSUB is used to pick out one or several of the data elements in a data record.
- FCPSUB is used to give new values to one or several of the data elements in a data record.
- FCGLINE writes part of a form to a memory buffer.
- FCWLTXt displays the leading texts of a form.
- FCRDFO redisplay a form.

5.12.1 CLEAR DATA RECORD FCCLREC

This routine is used to clear the data elements in the data record.

FCCLREC(data record, status)

Input parameter: data record, contains the data elements corresponding to the fields in the data record field set. Format: X(length of data record).

Output parameter: data record, returns the cleared data record.

Numeric fields are given the value zero. Text fields are filled with spaces.

Unused bytes between the data elements are filled with spaces.

5.12.2 CLEAR SUBRECORD FCCLSUB

This routine is used to clear a subset of the data elements in the data record.

FCCLSUB(field list, first occurrence number, last occurrence number,
data record, status)

Input parameters: field list, contains field names specifying the data elements to be cleared. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be cleared. Format: I.

data record, contains the data elements corresponding to the fields in the data record field set. Format: X(length of data record).

If the subset is set to contain all fields in the data record field set (field list="*"), this call is identical to the FCCLREC call.

Numeric fields are given the value zero. Text fields are filled with spaces.

Unused bytes between the data elements are filled with spaces.

5.12.3 WRITE DOTS IN FIELDS FCWDOTS

This function may be used to display dots in one or several of the fields in the data record field set.

FCWDOTS(field list, first occurrence number, last occurrence number,
status)

Input parameters: field list, is a list of field names and/or field group names specifying in which fieldsdots are to be written. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be used. Format: I.

For a new form the screen is cleared according to the "mode" parameter in the last call to FCDECFN.

For a new form the leading texts in the form are displayed if the "mode" parameter in the last call to FCDECFN is not equal to 2.

5.12.4 CLEAR FIELDS FCCLFDS

This routine is used to clear the fields on the screen.

FCCLFDS(mode,status)

Input parameters: mode: the parameter has three possible values:

- 0: Clear all fields in the form.
- 1: Clear all fields in the data record field set.
- 2: Clear the fields that were used in the last call to FCDECRC, FCESUB, FCWSUB, FCGSUB, FCWDOTS, FCPSUB, FCCLSUB, FCSETAT, FCSETMR or FCCLMR. (Calls using the "field list" parameter.)

Format: I.

For a new form the screen is cleared according to the "mode" parameter in the last call to FCDECFN.

For a new form the leading texts in the form are displayed if the "mode" parameter in the last call to FCDECFN is not equal to 2.

The routine clears the fields by displaying dots in the fields.

5.12.5 GET DATA ELEMENTS FROM A DATA RECORD FCGSUB

This routine is used to pick out one or several of the data elements in a data record.

FCGSUB(field list, first occurrence number, last occurrence number,
data record, sub data record, status)

Input parameters: field list, contains field names for all fields to be picked out. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be used. Format: I.

data record, contains the data elements corresponding to the fields in the data record field set. Format: X (length of data record).

Output parameter: sub data record, returns the data elements corresponding to the fields in the "field list". Format: X(length of sub data record).

5.12.6 PUT DATA ELEMENTS INTO A DATA RECORD FCPSUB

This routine is used to give new values to one or several of the data elements in a data record.

FCPSUB(field list, first occurrence number, last occurrence number,
data record, sub data record, status)

Input parameters: field list, contains field names for all fields to be updated. See p. 52 for the definition of the field list parameter. Format: X(length of field list).

first occurrence number and last occurrence number, are used if field names with no occurrence part are given. For these field names, all corresponding fields with occurrence numbers between (and including) "first and last occurrence number" will be used. Format: I.

data record, contains the data elements corresponding to the fields in the data record field set. Format: X(length of sub data record).

sub data record, contains data elements corresponding to the fields in the field list. Format: X(length of data record).

Output parameter: data record, returns the updated data record.

5.12.7 GET FORM LINE FCGLINE

This routine can be used to write part of a form with leading texts and fields to a memory buffer.

FCGLINE(from line, to line, data record, buffer area, status)

Input parameters: from line and to line: are line numbers relative to the start of the form. Format: I.

data record, is a buffer containing data elements corresponding to the fields in the data record field set. Format: X(length of data record).

Output parameters: buffer area, is a buffer where the form text should be stored. Format: X(length of buffer).

The text is stored as a continuous byte stream.

Use the FCFRMIN call to get the length of a line.

5.12.8 WRITE LEADING TEXTS FCWLTXT

Because of optimization, the leading texts will only be displayed in the first write/edit call after the form has been declared. If the screen image is for some reason destroyed, the leading texts may be rewritten using this call.

FCWLTXT(mode, status)

Input parameter: mode, the parameter has three possible values:

-1: No clear before displaying the leading text.

0: Clear the whole screen before displaying the leading texts.

1: Clear only the part of the screen used by the form before displaying the leading texts.

Format: I.

5.12.9 REDISPLAY FORM FCRDFO

This function may be used to redisplay a form.

FCRDFO(mode, data record, status)

Input parameters: mode, there are three possible values:

- 1: No clear before redisplaying the form.
- 0: Clear the whole screen before redisplaying the form.
- 1: Clear only the part of the screen used by the form before redisplaying it.

Format: I.

data record: is a buffer containing data elements corresponding to the fields in the data record field set. Format: X(length of data record).

5.13 FORM INDEPENDENT CALLS

The following routines are available:

- FCRTXT is used to read a text string from the terminal.
- FCWTXT is used to write a text string to the terminal.
- FCCLSCR is used to clear a rectangular area on the screen.
- FCPCUR is used to move the cursor to a given position on the screen.
- FCEBUF will enforce the emptying of the terminal output buffer (ND-500 only).
- FCRCHR is used to read one character without echo from the terminal.
- FCBELL is used to give a "beep" from the terminal.

5.13.1 READ TEXT FCRTXT

The routine is used to read a text string from the terminal from a given position.

FCRTXT(line, column, text, length of text, status)

Input parameters: line, is the start line. Format: I.

column, is the start column. Format: I.

length of text, is the maximum length in number of bytes of text string to be read. Format: I.

Output parameters: text, returns the text string read from the terminal. Format: X(length of text).

length of text, is the length in number of bytes of the text string actually read. Format: I.

The value -1 used in the parameters "line" or "column" means the last position, i.e., last column or line.

The "field" for entering the text is cleared (filled with spaces). The terminal operator may then enter a text which can contain any alphanumeric characters. The same editing facilities may be used as when editing a field. The reading is terminated when carriage return is pressed.

Editing is described in chapter 6.

5.13.2 WRITE TEXT FCWTEXT

This routine is used to display a text string on the terminal from a given position.

FCWTEXT(line, column, text, length of text, status)

Input parameters: line, is the start line. Format: I.

column, is the start column. Format: I.

text, contains the text string to be displayed.
Format: X(length of text).

length of text, is the length in number of bytes of text string to be displayed. Format: I.

The value -1 used in the parameters "line" or "column" means the last position, i.e., last column or line.

5.13.3 CLEAR RECTANGULAR AREA ON SCREEN FCCLSCR

The routine is used to clear a rectangular area on the screen.

FCCLSCR(from line, from column, to line, to column, status)

Input parameters: from line, clear from and including this line.
Format: I.

from column, clear from and including this column.
Format: I.

to line, clear to and including this line.
Format: I.

to column, clear to and including this column.
Format: I.

The value -1 used in the parameters means the last position, i.e., last column or line.

5.13.4 POSITION CURSOR FCPCUR

This routine may be used to position the cursor at a given position on the screen.

FCPCUR(line, column, status)

Input parameters: line, specifies the line position. Format: I.

column, specifies the column position. Format: I.

The value -1 used in the parameters "line" or "column" means the last position, i.e., last column or line.

5.13.5 EMPTY BUFFER *FCEBUF*

On the ND-500, because of optimization, leading texts, fields etc. written to the terminal using FOCUS calls, will be buffered. The text will be displayed on the terminal:

- on request for input using a FOCUS call
- when the buffer is full (the size is 2000 bytes)
- when FOCUS is terminated
- if the routine FCEBUF is called

Sometimes, for example, when using a HOLD function, it is necessary to enforce the emptying of this buffer. The FCEBUF call can then be used.

FCEBUF(status)

No action will be performed on the ND-100.

5.13.6 READ CHARACTER *FCRCHR*

This routine may be used to read one character without echo from the terminal.

FCRCHR(input character, status)

Output parameter: input character, returns the key value of the input character. See p. 53 for the definition of key value. Format: I.

5.13.7 BELL *FCBELL*

This routine may be used to give a "beep" from the terminal.

FCBELL(status)

CHAPTER 6

TERMINAL OPERATOR INTERFACE

6 TERMINAL OPERATOR INTERFACE

The terminal operator is the person who enters data by typing it in on a terminal. Every time the user application performs an edit call, control is transferred to the terminal operator. S/he may then enter data in the fields of the current form, or change the data already displayed. When the terminal operator has finished with the current form, s/he terminates the editing, and control is again transferred to the user application.

In FOCUS there is a set of break characters (function keys). When these are typed, no characters are echoed and some kind of action is taken. Illegal characters which are not break characters result in a "beep" and no other action.

The following chapters describe the default navigation strategy used in FOCUS, and which terminal keys are connected to the editing functions and functions like help, redisplay, cancel, etc. Some of these functions are break functions (see section 5.9.2), in which case the function number is added to the description (ex. Brf: 1000).

The section Define Terminal Operator Interface (5.9) describes how the programmer may redefine these settings. It is not, however, possible to connect functions that are not break functions to other function keys. The function keys in use for these functions may nevertheless be connected to other break functions.

6.1 EDITING A FIELD

All fields are edited as left-justified strings. This means that right-justified fields will be redisplayed as left-justified when the terminal operator enters the field, and redisplayed as right-justified when s/he leaves the field again.

Available editing functions:

"right arrow": Move the cursor one character position to the right. If the current cursor position is the last character position in the field, the cursor will move to the first position in the next field on the current line. If the current field is the last field on the line, a "beep" is given.

"left arrow" : Move the cursor one character position to the left. If the current cursor position is the first character position in the field, the cursor will move to the first position in the previous field on the current line. If the current field is the first field on the line, a "beep" is given.

<A> : Delete the character under the cursor or the previous character if the cursor is immediately beyond the last significant character in the field. (The notation <A> means the CTRL key plus the character A.)

a : Same as <A>.

<F> "x" : Move the cursor right to the next occurrence of "x" within the field.

If "x" equals $\leftarrow \downarrow$, the cursor is moved to the last position in the field.

If "x" equals <F>, the cursor is moved to the position after the last significant character in the field. If this would cause the cursor to be positioned outside the field, it will instead be placed in the last position in the field.

If "x" is not found, a "beep" is given.

<R> "x" : Move the cursor left to the first occurrence of "x" within the field.

If "x" equals $\leftarrow \downarrow$, the cursor is placed at the start of the field.

If "x" equals <R>, the cursor is placed under the first non-blank character in the field.

If "x" is not found, a "beep" is given.

<D> "x" : Delete characters in the field up to and including the character "x".

If "x" equals <D>, the entire field is cleared. If the field is a line in a text field, the lines (i.e. field occurrences) following this line will be moved one line upwards. The last line will be cleared. (Brf: 2009).

If "x" equals $\leftarrow \downarrow$, the rest of the field from the current position to the end of the field is cleared.

If "x" equals <R>, the characters from the start of the field up to but not including the cursor position will be deleted.

If "x" is not found, a "beep" is given.

<E> : Set/reset expand mode.

FUNC A : Set/reset alternative character set mode.

F1 : Same as <D><D>.

F2 : All lines (i.e. field occurrences) from and including this line will be moved one line downwards and a new blank line will be inserted. The function is only performed if the field is a text field. (Brf: 2007).

<L> : Same as F2.

6.2 NAVIGATING IN A FORM

↵ : Move the cursor to the first position in the next field. If the current field is the last field in the edit set, control is given to the user program. (Brf: 1000).

===> : Move the cursor to the first position in the next field. If the current field is the last field in the edit set, control is given to the user program. (Brf: 1000).

<=== : Move the cursor to the first position in the previous field. If the current field is the first field in the edit set, control is given to the user program. (Brf: 1001).

"right arrow": given in the last position of a field : Move the cursor to the next field on the current line. If the current field is the last field on the line, a "beep" is given. (Brf: 1003).

- "left arrow" : given in the first position of a field : Move to previous field on the current line. If the current field is the first field on the line, a "beep" is given. (Brf: 1002).
- "up arrow" : The cursor is moved to the first position in the field above. If the field is a text field, the cursor is moved to the position immediately above. If there is no field immediately above, the cursor is moved to the nearest field to the right on the line above. If there are no fields to the right, the cursor is moved to the nearest field to the left. If there are no fields at all on this line, the line above this will be tried. If there are no fields on the lines above, a "beep" is given. (Brf: 1004).
- "down arrow" : The cursor is moved to the first position in the field below. If the field is a text field, the cursor is moved to the position immediately below. If there is no field immediately below, the cursor is moved to the nearest field to the right on the line below. If there are no fields to the right, the cursor is moved to the nearest field to the left. If there are no fields at all on this line, the line below it will be tried. If there are no fields on the lines below, a "beep" is given. (Brf: 1005).

6.3 TERMINATING FORM EDITING

There are several possible ways in which to terminate the editing of a form:

- ↵ given in the last field in the edit set : Normal termination; data elements are transferred to the user application, and the user application is given control. (Brf: 0).
- <=== given in the first field in the edit set : Normal termination. Data elements are transferred to the user application, and the user application is given control. (Brf: 2).

- ==> given in the last field in the edit set : Normal termination; data elements are transferred to the user application, and the user application is given control. (Brf: 0)
- FUNC "home" : Normal termination. Data elements are transferred to the user application. (Brf: 1)
- FUNC <W> : Error termination. The user application is given control. The data elements for the current field are not transferred to the user application. (Brf:-1)

6.4 OTHER FUNCTIONS

- HELP : The help picture for the current field is displayed. Any character typed now will redisplay current form and "saved" forms, and position the cursor in the current field. (Brf: 2002)
- FUNC @ : The current form and "saved" forms are redisplayed. (Brf: 2003).
- CANCEL : The content of the field when the field was entered is restored. (Brf: 2004).
- COPY : The previous occurrence of the current field is copied to the field. Next field is entered. If the current field has occurrence number one, a "beep" is given. (Brf: 2005).

CHAPTER 7

LOADED FORMS

7 **LOADED FORMS**

Form descriptions can be loaded with the program or on a segment. They can then be used directly in the program without having to be read from a file.

To be able to do this, a relocatable file containing the form descriptions must be generated. This is done using the program FOCUS-COMPIL:PROG. The user must specify what forms are to be included, and whether the file is to be an ND-100 1-bank, ND-100 2-bank or an ND-500 version. This file must be loaded before the FOCUS-RTS is loaded.

Programs using loaded forms can also use forms on files.

Using FOCUS-COMPIL

When starting FOCUS-COMPIL, a form will appear, and you can fill in the following: the name and format of the relocatable file, the form files from which the forms are to be read, and the names of these forms. When you have finished the editing, push the PRINT key to compile the forms. Use the EXIT key to leave the program.

Form names may be truncated using an asterix (*). E.g., entering HELP* as a field name will cause all forms on the file which begin with HELP to be compiled.

Below is an example of how to fill in this form. The information which has been filled in, is underlined.

PROGRAM TO COMPILE FORMS IN FOCUS
Push the HELP key or ? whenever information is needed

Relocatable format? (1-BANK / 2-BANK / NRF) : 1-BANK
Name of relocatable file: "COMPILED-FORMS"

No.	Form file name	Form name
1	<u>FORMFILE-A</u>	<u>FORM1</u>
2		<u>FORM2</u>
3		<u>HELPPFORM</u>
4	<u>FORMFILE-B</u>	<u>*</u>
5		
6		
7		
8		
9		
10		

Push the PRINT key to compile the forms; EXIT to leave the program.

If you press the PRINT key now, a relocatable file, COMPILED-FORMS:BRF (for ND-100, 1-bank), will be created. It will contain the forms FORM1, FORM2 and HELPPFORM from the form-file FORMFILE-A, and all forms from the form-file FORMFILE-B.

When specifying form file names and form names, lines can be inserted/deleted using the CTRL-L/CTRL-W keys. Using a down arrow in the last line or an up arrow in the first line of the form will cause the form to scroll. Up to 180 form names may be specified, and up to 250 forms may be compiled simultaneously.

It is not possible to run FOCUS-COMPIL-G as a mode file.

C H A P T E R 8

USER DEFINED CONTROL

8 USER DEFINED CONTROL

During the field definition various control algorithms can be connected to the field. FOCUS offers a set of standard control algorithms, for example, legal/illegal value checks and legal date checks. The user may also write his/her own control algorithms.

A subroutine with the name FCUCONT will be called if the parameter "FCUCONT par." is given a value during the field definition. The user control routine can be called when entering a field and/or when the editing of a field is terminated. The routine can be written in FORTRAN, COBOL or PLANC and must have the following format:

FCUCONT(parameter string, field name, occurrence number, data record,
edit info, continue flag, changed flag, display flag, status)

Input parameters: parameter string, contains the string which was given as parameter when the user control was defined. This could, for example, be a name specifying which part of the user control routine is to be performed. Format: X(length of string). NOTE that trailing spaces are stripped.

field name, contains the name of the field from which the user routine was called. Format: X(length of field name).

occurrence number, contains the occurrence number of the field from which the user routine was called. Format: I.

data record, contains the data record. Format: X(length of data record).

edit info, contains 4 elements:

Element 1: Termination code.

Element 2: Termination character.

Element 3: Number of significant characters in edit buffer.

Element 4: Accept-value flag is set to 1 if the accept-value function is given, otherwise the flag is set to 0.

Format: I(4).

changed flag, is an integer with two possible values:

- 0: The field has not been changed after it was entered.
- 1: The field has been changed after it was entered.

Format: I.

Output parameters: data record, (the data record may have been changed in the user routine.)

continue flag, must be set to -1 if the editing is to continue in the same field. Format: I.

changed flag, must be set to 1 if the data element has been changed in the user routine.

display flag, must be set to 1 if changes are made to the field that require a redisplay of the field. Format: I.

status, status >< 0 will lead to an immediate return to the user program. The status will be returned to the user program as the status from the edit call.

The user control routine FCUCONT must be loaded before the FOCUS library is loaded.

To find the position of the current data element in the data record, use the FCFLDIN call.

The following FOCUS routines cannot be called from FCUFUNC:

FCCLFO	FCDECF	FCDECFN	FCDECRC	FCDESB	FCEFLD
FCEREC	FCESUB	FCINITE	FCPOSFO	FCSAVFO	

A P P E N D I X A

Error Codes from FOCUS-RTS

Error codes which are less than 1000 decimal are SINTRAN III error codes and can be found in the SINTRAN III Reference Manual. For error codes from the ND-500 monitor, see the ND-500 documentation.

No. octal decimal Meaning:

10401b	4353	Illegal parameter given to FCINITE.
10402b	4354	FCINITE not called.
10404b	4356	No such form name.
10405b	4357	Form buffer too small for this form.
10406b	4358	Form name not declared.
10407b	4359	Illegal occurrence number.
10411b	4361	No such field name.
10412b	4362	Illegal field number.
10413b	4363	Message not terminated by an apostrophe.
10414b	4364	Illegal screen position.
10415b	4365	Field not in data record field set.
10416b	4366	Field not in edit set.
10417b	4367	Illegal character in field name.
10420b	4368	Cursor position outside field.
10421b	4369	Device number is not a terminal.
10422b	4370	Illegal "edit-mode" to FCESUB.
10423b	4371	Illegal "mode" to FCDECFN.
10424b	4372	Illegal "edit-mode" to FCEREC.
10425b	4373	Illegal "edit-mode" to FCEFLD.
10426b	4374	Illegal "mode" to FCCLFDS.
10427b	4375	Too many fields specified.
10430b	4376	Illegal segment number.
10431b	4377	Illegal "length" to FCWFLD.
10433b	4379	Illegal key value.
10434b	4380	Illegal break function.
10435b	4381	Illegal "number" to FCFLDBR.
10436b	4382	Illegal "number" to FCFRMBR.
10437b	4383	Break function loop.
10440b	4384	Illegal "mode" to FCWLTXt.
10441b	4385	Illegal form file type.
10442b	4386	Illegal "length" to FCSCRIN.
10443b	4387	Illegal "length" to FCFLDIN.
10444b	4388	Illegal "length" to FCFLDST.
10445b	4389	User control routine not loaded.
10446b	4390	User function not loaded.
10447b	4391	No fields in the edit set.
10450b	4392	Illegal "length" to FCFRMIN.
10451b	4393	This function needs a current field.
10452b	4394	This function cannot be performed from the cursor position.
10453b	4395	Form positioned outside screen.
10454b	4396	Illegal "line" to FCGLINE.
10455b	4397	Illegal "mode" to FCRDFO.
10456b	4398	Too many forms used simultaneously.
10457b	4399	This feature has not been loaded.
10501b	4417	Form description is not up to date.
10502b	4418	File is not a form file.
10503b	4419	Bad linking (not user error).

Error codes from VTM (Virtual Terminal Manager).

<u>No. octal</u>	<u>decimal</u>	<u>Meaning:</u>
10030b	4120	"X" value (i.e., column) out of range.
10031b	4121	"Y" value (i.e., line) out of range.
10032b	4122	Character outside legal range for this terminal type.
10033b	4123	The function or character to be output does not have an entry in the DDB table and cannot be simulated.
10034B	4124	No embedded routine exists with specified number.
10035B	4125	No attributes routine exists with specified number.
10036B	4126	Cannot perform this function because the current "X" position is unknown.
10037B	4127	Cannot perform this function because the current "Y" position is unknown.
10040B	4128	Cannot position relatively because current cursor position is unknown.
10060B	4144	No cursor positioning routine with the specified number can be found (forgotten to load "VTM-CPOS:BRF" and/or "VTM-CPAR:BRF"?).
10062B	4146	The function needed for "vtcpo" (cursor left, right, up, down, or <cr>) is not available on this terminal, so positioning is impossible.
10063B	4147	Cursor left function not available for "vtspbk".
10064B	4148	More than "max. chars" characters read by "mn310".
10065B	4149	Return from "mn310" when no break condition.
10077B	4159	Unknown terminal type.
10150B	4200	Terminaltype in DDB file inconsistent with file name (VTM-list).
10151B	4201	The string length of a sequence possibly exceeds 127 characters, thus looking like a sequence formula request (VTM-list).

A P P E N D I X B

Terminal Information

Standard FOCUS and Mini FOCUS

The standard version of FOCUS can be used on all terminals for which VTM-tables are generated. A 'mini' version has been made to save some space, but it can be run on a limited number of terminals only, and on some of the terminals it will have limited functionality.

Some ND-supported terminals and the corresponding terminal types are listed below:

Tandberg-TDV2115-Standard	(type 3)
Hazeltine-1520	(type 10)
Tandberg-TDV2215-Extended	(type 36)
Tandberg-TDV2215-SDS-V2	(type 52)
Tandberg-TDV2200/9-ND-NOTIS	(type 53)
Facit-4420-ND-NOTIS	(type 57)
Tandberg-TDV2200/9-V2-ND-NOTIS	(type 83)
Facit-twist (24-lines mode)	(type 91)
Facit-twist (72-lines mode)	(type 92)
Tandberg-TDV2200/9S-ND-NOTIS	(type 93)

The terminal tables that are found on the release diskettes are used by Standard FOCUS. They support the terminal types:

3, 36, 52, 53, 57, 83, 91, 92 and 93

Mini FOCUS supports the following terminal types:

3, 10, 36, 52, 53, 57, 83, 91, 92 and 93

Input sequences

Most ND-supported terminals will generate an input sequence for some terminal keys to facilitate more than 128 input characters. These sequences will most often contain the escape character. This is the reason why FCINIT disables escape on these terminals.

The terminals which do not have these special keys can generate the same input by using the FUNC key (or CTRL + underline) followed by some other character. For instance FUNC + ? will work in the same way as the HELP key. See "NOTIS reference card for non-NOTIS terminals" (ND-63.031Q.01, enclosed with this manual) for detailed information about the equivalence between special keys and the FUNC + char. input.

Display Attributes and Alternative Character Sets

FOCUS can handle display attributes (inverse video, blink etc.) and combinations of these both with leading texts and in the fields FOCUS can also handle the full first alternative character set (corresponding to Greek + graphic mode in NOTIS-WP). The table below shows which of these features that are available on some ND-supported terminals. Most other terminals do not support these features. (S = Standard FOCUS, M = Mini FOCUS):

Terminal types	36, 52	53, 83, 93	57	91, 92
High intensity				SM
Low intensity	SM	SM	SM	
Inverse video	SM	SM	SM	SM
Blink	SM	SM	SM	SM
Underline	SM	SM	SM	SM
Invisible	SM	SM		
Max. number of combined attrib.	3	3	2-3	4
Graphic chars.	SM	SM	SM	SM
Greek chars.		SM		S

Special Cursor Positioning Routines

Some terminals require cursor positioning routines that are not included in the FOCUS libraries for ND-100. If you get the error status code 4144 (decimal) when trying to run FOCUS from a terminal, load the routines VTM-CPOS:BRF and VTM-CPAR:BRF if ND-100 1-bank (VTM-2B-CPOS:BRF and VTM-2B-CPAR:BRF if ND-100 2-bank) before loading the FOCUS library.

These routines are found on the release diskettes.

A P P E N D I X C

Converting from Older FOCUS Versions to the G Version

Converting from the E version to the G version

Use the following procedure:

- Enter FOCUS-DEF-G and use the command MAKE-UPTODATE-FORM for each form file that should be converted.
- Make the necessary changes in the application program. See below.
- Load the application program with the G version of the library. Forms loaded with the program must be re-compiled using FOCUS-COMPIL-G.

Changes

- "Must read" is now also working if edit mode = 0 (modify).
- Two parameters, field name and occurrence number, are added to the FCUFUNC routine.
- Display attribute handling in FOCUS-DEF: SHIFT aaa is used instead of F3.
- The DESCRIBE-FORMS command in FOCUS-DEF: the user interface and the layout of the information is changed.

Additions

The following new features are implemented (see the relevant chapters for further details):

- Form info calls. Return the x- and y-coordinates and size of a form, the number of fields and the field names.
- New Start-Edit-in-Field routine and Execute-Break-Function routine. These two routines will make navigating between forms and program defined navigation between the fields in a form easier.
- Calls to handle more than one form simultaneously.
- Call to enforce the emptying of the terminal output-buffer on the ND-500.
- New call for displaying 'dots' in the fields on the screen. A list of field names can be specified.
- User defined 'dot'.
- Accept value. The field value checks will be overruled.

- Uppercase conversion fields. The characters will be converted to uppercase before being stored in the data element, and the alphabetic range can be specified.
- New version of FOCUS-COMPIL. A maximum of 250 forms can be compiled, and 'compile all forms in a file' is possible.
- Text fields. With word-wrap and insert/delete line.
- Blank when zero. Corresponds to the COBOL feature. Works for numeric fields and string-fields.
- Storage code REAL4, REAL6, REAL8.
- The field occurrence number range can be specified to be line oriented or column oriented.
- Dump form, i.e., dump lines of a form, with leading texts and field contents, to a memory-buffer.
- New element in the 'edit status buffer': Cursor position in a field.
- New element in 'edit info' array in the FCUCONT routine: 'Accept value' flag.
- FOCUS is made more modular. It is now possible to replace some functions by dummy-routines to save memory space.
- Optimization on output. Writing to a terminal will use less CPU-resources.

Converting from the C/D version to the G version

To go from the C/D version to the G version, follow the following procedure:

- Change the application program according to the changes specified above and below.
- Convert the form file to the format used by the E version by using FOCUS-CONVRT-G. See below.
- Enter FOCUS-DEF-G and use the command MAKE-UPTODATE-FORMS for each converted form file.
- Load the application program with the G version of the library.

Changes from the FOCUS-C/D versions to the E version

The following changes have been made:

- Navigate mode has been removed.
- The call interface to FCDECRC is changed. New format:

FCDECRC(sublist, first-occurrence-number, last-occurrence-number,
status)

(Note that field names are now separated by spaces instead of
apostrophes. The apostrophe is now used to terminate the string.)

- The parameter "data element length" in the call FCEFLD is an output
parameter only.
- Form names may be terminated in the same way as form files, i.e.
by using an apostrophe.
- The default and only legal form file type is :FORM.
- The syntax of a field name is restricted.
- Names of fields, forms and files can be given in uppercase or
lowercase when given as parameters.
- Occurrence numbers given with field names can be from one to three
digits.
- Messages written with FCGMSGE or FCZMSGE will be cleared by FOCUS
when leaving a field in an edit call.
- The format of the FOCUS form files has been changed. This means
that forms used with the C/D version must be converted before they
can be used with the E version.

Convert old form files to FOCUS-E form files

Enter FOCUS-DEF-G. Create and initialize a new form file to contain
your converted old forms (command 'CREATE-FILE'). The new file format
consists of a compacted runtime format and a definition format. This
will require somewhat more space, so you should probably allocate more
pages for your new form file than what was used for the old one.

Start the program FOCUS-CONVRT-G, which will ask four questions (you
may at any time abort the program by pressing the EXIT key):

- the name of your old form file
- the name of the new file created by FOCUS-DEF
- an output file for the printout from the program. This file
contains the names of forms and possible warnings or errors.
Default is terminal.
- whether a comma or a dot is to be used as decimal point for numeric
fields containing decimals.

If any of the following conditions concerning fields occur, the program
will give a warning together with the name of the field.

- Illegal characters in the old field name. (The name will not be changed, but the field must later be renamed using FOCUS-DEF.)
- Fill character in a left justified numeric field is not a space. Fill character is converted to space.
- Illegal fill character in a right justified numeric field. Legal fill characters are ' ' (COBOL-format: Z), '0' (COBOL-format: 9) and '*' (COBOL-format: *). Illegal fill characters are converted to '*'.
- A floating decimal point numeric field or a left justified numeric field is found. These are not allowed by FOCUS-E. The field is converted to a text field with digits and decimal point as legal characters. Any fill character which is not a space, is reset to space.

A P P E N D I X D

Loading FOCUS Applications

The following files should be loaded together with the application program:

ND-100 1-bank ordinary version:

VTM-1B-ARRAY-E:BRF	(1) (see below)
VTM-CPOS-F:BRF	(2) (see below)
VTM-CPAR-F:BRF	(2) (see below)
FOCUS-DUMMY-G:BRF	(4) (see below)
FOCUS-CODE-G:BRF	
FOCUS-DATA-G:BRF	

ND-100 1-bank version using Mini-VTM: (3) (see below)

FOCUS-DUMMY-G:BRF	(4) (see below)
MFOCUS-CODE-G:BRF	
MFOCUS-DATA-G:BRF	

ND-100 2-bank ordinary version:

VTM-2B-ARRAY-E:BRF	(1) (see below)
VTM-2B-CPOS-F:BRF	(2) (see below)
VTM-2B-CPAR-F:BRF	(2) (see below)
FOCUS-2DUMMY-G:BRF	(4) (see below)
FOCUS-2CODE-G:BRF	
FOCUS-2DATA-G:BRF	

ND-100 2-bank version using Mini-VTM: (3) (see below)

FOCUS-2DUMMY-G:BRF	(4) (see below)
MFOCUS-2CODE-G:BRF	
MFOCUS-2DATA-G:BRF	

ND-100 TPS version using Mini-VTM: (3) (see below)

FOCUS-DUMMY-G:BRF	(4) (see below)
FOCUS-TPCODE-G:BRF	
FOCUS-TPDATA-G:BRF	
PLANC-1BANK-E:BRF	

ND-500 version:

VTM-ARRAY-E:NRF	(1) (see below)
FOCUS-CODE-G:NRF	
FOCUS-DATA-G:NRF	

- (1) This file contains the terminal dependent data and can be loaded with the program to avoid the file DDBTABLES-E:VTM being opened or read from. The file must be loaded prior to the FOCUS library.
- (2) These files contain special cursor positioning routines needed for some terminal types, and should be loaded with the program if you get error status 4144 (decimal) when running FOCUS. The files must be loaded prior to the FOCUS library.
- (3) Refer to Appendix B for information on when the FOCUS libraries using Mini-VTM can be used.
- (4) This file contains dummy routines that can be used to prevent code for used features from being loaded, thereby saving memory space. See below.

Removing unused features from the FOCUS runtime library.

To save memory space, it is possible to remove some unused features in the library by replacing them with dummy routines. This is done by referencing the features to be removed, and then loading the file containing the dummy routines. The following features can be removed:

<u>Symbol</u>	<u>Feature</u>
FCBCD	Fields with storage type BCD.
FCINTEGER	Fields with storage types INT2 and INT4.
FCREAL	Fields with storage types REAL4, REAL6 and REAL8.
FCTEXT	Text fields (with the functions word-wrap, insert line and delete line).
FCDATE	Check for legal date.
FCPERSONNUMBER	Check for legal person-number. (Identity number) (According to the Norwegian standard.)
FCBANKACCOUNT	Check for legal bank-account number. (According to the Norwegian standard.)
FCPOSTALACCOUNT	Check for legal postal-account number. (According to the Norwegian standard.)
FCVALUECHECK	Legal/illegal value checks.
FCREADFORM	Read a form from a form file. The dummy-routine can be used if only loaded forms are used.

Dummy routines are only available on the ND-100.

Examples

Example of loading a COBOL application with a 2-bank library on a ND-100. Fields with storage code REALx and person-number check are not used, and are replaced by dummy routines. The file LOADED-FORMS contains forms to be loaded with the program, and the file UCONT contains the user control routine.

```
@NRL  
PROGRAM-FILE <name-of-user-application>:PROG  
SIZE 2000  
LOAD <main-program>  
LOAD LOADED-FORMS  
LOAD UCONT  
REFERENCE FCREAL,,,  
REFERENCE FCPERSONNUMBER,,,  
LOAD FOCUS-2DUMMY  
LOAD FOCUS-2CODE-G  
LOAD FOCUS-2DATA-G  
LOAD COBOL-2BANK  
EXIT
```

Example of loading a FORTRAN application using FOCUS on a ND-500.

```
@ND-500 LINKAGE-LOADER  
ABORT-BATCH-ON-ERROR OFF  
RELEASE-DOMAIN <name of application>  
DELETE-DOMAIN <name of application>  
ABORT-BATCH-ON-ERROR ON  
SET-DOMAIN "<name of application>"  
LOAD <main program>:NRF  
LOAD VTM-ARRAY-E  
LOAD FOCUS-CODE-G  
LOAD FOCUS-DATA-G  
LOAD FORTRAN-LIB  
LOAD EXCEPTION-LIB  
EXIT
```

In FORTRAN versions older than the F version, the EXCEPTION-LIB is included in FORTRAN-LIB, so it should not be loaded explicitly as it was in this example.

If COBOL is used, COBOL-LIB must be the last file to be loaded.

If using a FOCUS library that uses Mini-VTM together with ACCEPT/DISPLAY in COBOL, COBOL versions H or later must be used.

A P P E N D I X E

A Few Examples of Application Programs Using FOCUS

A Few Examples of Application Programs Using FOCUS

The examples 1, 2, and 3 are written in FORTRAN. They all use a form named MY-FORM which has been defined in FOCUS-DEF, and stored on the file TEST:FORM.

The form has the following layout:

```
=====
Name: ..... Date: .....
Addr: .....
No:   Descr.                      Weight   Cost
-----
.... .....                      .....
.... .....                      .....
.... .....                      .....
.... .....                      .....
.... .....                      .....
.....
=====
```

The first three fields in the form have the names NAME, DATE and ADDR. The 'middle' part of the form consists of 4 fields called D1, D2, D3 and D4. Each of these fields has five occurrences, so that a description of five 'parts' can be written in a uniform way in the form. The lower two 'result' fields are called R1 and R2.

The fields NAME, DATE, ADDR and D2 are text fields. The DATE field is defined with the system control \$DATE, so that a check for legal date will be performed. The D1, D3, D4, R1 and R2 fields are numeric fields, the first three with Integer 2 storage, the last two with Integer 4 storage.

A group named PART has been defined (in FOCUS-DEF), consisting of the fields D1, D2, D3 and D4.

In the second example, user control routines are connected to the fields D3 and D4. The second 'FCUCONT par.' parameter in the FOCUS-DEF field definition is filled in with the letters WE for the D3 fields and CO for the D4 fields.

EXAMPLE 1

This program will edit all fields in the form except the R1 and R2 fields. The sums of all occurrences of D3 and D4 will be written to the R1 and R2 fields after the editing of the form is completed.

```
PROGRAM FCTEST1

C    >> Form file name
      CHARACTER * 20 FORMFILE
      INTEGER IFORMFILE(10)
      EQUIVALENCE (FORMFILE,IFORMFILE)

C    >> Form name
      CHARACTER * 8 FORMNAME
      INTEGER IFORMNAME(4)
      EQUIVALENCE (FORMNAME,IFORMNAME)

C    >> Data record
      INTEGER IDATARECORD(200)

C    >> Variables to contain total weight and total cost of 'parts'
      DOUBLE INTEGER TOTALWEIGHT,TOTALCOST

C    >> FORTRAN FUNCTIONS to sum up occurrences
      DOUBLE INTEGER SUMWEIGHT,SUMCOST

      INTEGER I,MODE,EDITMODE,STATUS,ERRSTAT

      INTEGER INITARR(10),FORMBUFF(2000)

C    >> Initiate array for FCINITE - call
      INITARR(1) = 4000
      INITARR(2) = 1
      INITARR(3) = 0
      INITARR(4) = 0
      INITARR(5) = 2
      INITARR(6) = 0
      INITARR(7) = 0
      INITARR(8) = 0
      INITARR(9) = 0
      INITARR(10) = 0

C    >> Initiate FOCUS
      CALL FCINITE (INITARR,FORMBUFF,STATUS)
      IF (STATUS .NE. 0) GO TO 999

C    >> Tell FOCUS which form file to use
      FORMFILE = 'TEST'
      CALL FCDECF (IFORMFILE,STATUS)
      IF (STATUS .NE. 0) GO TO 999
```



```
C      >> Clear screen before editing  
MODE = 0  
  
C      >> Read form description from file  
FORMNAME = 'MY-FORM'''  
CALL FCDECFN (IFORMNAME,MODE,STATUS)  
IF (STATUS .NE. 0) GO TO 999  
  
C      >> Normal mode for editing  
EDITMODE = 1  
  
C      >> Declare record to contain all fields except  
C      >> those named R1 and R2.  
CALL FCDECRC ("- R1 R2'",1,1,STATUS)  
IF (STATUS .NE. 0) GO TO 999  
  
C      >> Edit the defined record  
CALL FCEREC (IDATARECORD,EDITMODE,STATUS)  
IF (STATUS .NE. 0) GO TO 999  
  
C      >> Call FUNCTIONS for summing weights and costs  
TOTALWEIGHT = SUMWEIGHT (IDATARECORD,STATUS)  
IF (STATUS .NE. 0) GO TO 999  
TOTALCOST   = SUMCOST    (IDATARECORD,STATUS)  
IF (STATUS .NE. 0) GO TO 999  
  
C      >> Write sums to screen  
CALL FCWFLD ("R1'",1,TOTALWEIGHT,4,STATUS)  
IF (STATUS .NE. 0) GO TO 999  
CALL FCWFLD ("R2'",1,TOTALCOST,4,STATUS)  
IF (STATUS .NE. 0) GO TO 999  
  
C      >> Terminate FOCUS  
999 INITARR(2) = 0  
ERRSTAT = STATUS  
CALL FCINITE (INITARR,FORMBUFF,STATUS)  
C      >> If some error should have occurred:  
IF (ERRSTAT .NE. 0) WRITE (1,*) ' Error number: ',ERRSTAT  
  
STOP  
  
END  
  
C      >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  
C      >> FUNCTION for summing weights  
  
FUNCTION SUMWEIGHT (IDATARECORD,STATUS)  
  
DOUBLE INTEGER SUMWEIGHT  
INTEGER IDATARECORD,STATUS,I,SUMBUFF(5)  
  
C      >> Place all 5 occurrences of field D3 into SUMBUFF  
CALL FCGSUB ("D3'",1,5,IDATARECORD,SUMBUFF,STATUS)
```

[illegible]

EXAMPLE 2

This program shows the use of the FCUCONT routine. The program will edit all fields except R1 and R2. Each time editing has been performed on any occurrence of the D3 or D4 fields, the sum of all D3 or D4 occurrences (R1 and R2) will be updated and written to the screen.

The checks for error return (STATUS > 0) are not listed here.

PROGRAM FCTEST2

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Declarations as for previous program

Upstart as for previous program, FOCUS routines called:
FCINITE - FCDECFE - FCDECFN

```
C      >> Edit NAME, DATE and ADDR fields plus the PART group that
C      >> was defined in FOCUS-DEF (consisting of fields D1, D2,
C      >> D3 and D4).
      CALL FCESUB ("NAME.1 DATE.1 ADDR.1 PART'",
+                1,5, IDATARECORD, EDITMODE, STATUS)
```

Year	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Entity	Market	Interest	Debt	Assets	Equity	1 Year	Revenue	EBITDA	CapEx	Divid	Rating	Ref. No.
--------	--------	----------	------	--------	--------	--------	---------	--------	-------	-------	--------	----------

[illegible]

```

SUBROUTINE FCUCONT (PARAMETER,FLDNAME,OCC,IDATARECORD,
+                   EDITINFO,CONT,CHANGED,DISP,STATUS)

```

```

INTEGER PARAMETER,FLDNAME,OCC,IDATARECORD,EDITINFO,
+      CONT,CHANGED,DISP,STATUS
DOUBLE INTEGER SUMWEIGHT,SUMCOST,TOTALWEIGHT,TOTALCOST

```

CALL FCWSUB ("@" , 1, 1, IDATARECORD, STATUS)

[illegible]

EXAMPLE 3

This program shows the use of FCCHRRBR and the user defined routine FCUFUNC. The program will edit one form in the 'default' way, but with the following additions:

- If the CANCEL (FUNC X) or DELETE (FUNC D) key is pressed, a return to the user program will occur; any action may be performed in these cases.
- If the F6 (FUNC + L) or SHIFT F6 (FUNC + U) key is pressed when editing a field, a conversion to lowercase or uppercase characters is performed from current cursor position until the end of the field.

PROGRAM FCTEST3

Declarations as for previous program with the following additions:

INTEGER KEYS(13),FNCTIONS(13),EDSTAT(2)

C >> Keys which will cause a break function to be performed:
C >> All the predefined ones plus DELETE,F6
C >> and SHIFT F6.

DATA KEYS/15B,316B,302B,235B,227B,277B,300B,330B,27B,303B,304B,
+ 314B,325B/

C >> Break functions to be performed:
C >> They are the same as the predefined ones, except for
C >> - 330B (CANCEL) - controlled return with termination code 3
C >> This overrides the predefined function
C >> - 304B (DELETE) - controlled return with termination code 4
C >> No predefined function
C >> - 314B (F6) - will cause user function to be performed
C >> (Convert to lowercase) - No predefined function
C >> - 325B (SHIFT F6) - will cause user function to be performed
C >> (Convert to uppercase) - No predefined function

DATA FNCTIONS/1000,1000,1001,1,-1,2002,2003,3,2004,2005,4,
+ 9000,9001/

Upstart as for previous program, FOCUS routines called:
FCINITE - FCDECFF - FCDECFN

C >> Define function keys. Use the arrays given initial
C >> values in DATA statement.
CALL FCCHRRBR (13,KEYS,FNCTIONS,STATUS)

C >> Define an edit status buffer to contain termination code
C >> and termination character.
CALL FCDESB (EDSTAT,2,STATUS)

Norsk Data ND-60.137.5 EN

```

        ENDDO
ELSE IF (FUNCNO .EQ. 9001) THEN
C      >> SHIFT + F6 key - convert to uppercase;
C      >> from cursor position until last significant
      DO FOR I = CURPOS,SIGNIF
        IF (BUFFER(I:I) .GE. 'a' .AND. BUFFER(I:I) .LE. 'z') THEN
C          >> Indicate change of buffer - new display required
          CHANGED = DISP = 1
C          >> Get integer value of character, subtract 32
C          >> and convert back to character
          BUFFER(I:I) = CHAR(ICHAR(BUFFER(I:I)) - 32)
        ENDIF
      ENDDO
    ENDIF
  ENDIF

  IF (CHANGED .EQ. 1) THEN
C    >> Buffer has been changed in FCUFUNC routine
C    >> Copy local buffer back to edit buffer
    DO FOR I = 1,LENGTH
      EDITBUFF (I) = IBUFFER (I)
    ENDDO
  ENDIF

END

```

EXAMPLE 4

This is an example of how to use FOCUS from a COBOL program. It shows how to initiate FOCUS, how to define an edit status buffer, the definition of function keys and 'leaving field situations'.

Then an example of how to make a message system is shown. The messages are found on the forms MSG1 and MSG2 on the form file TEST. The messages are in fact leading texts, one line for each message. A message consists of a key of eight characters and the message text of 72 characters.

The routine INIT-MSG reads the messages into a message buffer. The routine WRITE-MSG finds the message corresponding to a message key, and displays the message on the screen.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE-4.
AUTHOR. NORSK DATA A/S.
DATA DIVISION.
WORKING-STORAGE SECTION.

```

```

* Data record
01 DATARECORD                PIC X(300).

01 ISTATUS                   COMP.
01 ERRLAB                    PIC X(10).
01 I                          COMP.
01 L                          COMP.

```

```
01 FORMBUFF          PIC X(2000).  
01 ERMSG             PIC X(80).
```

* Initiation array.

```
01 INITARR.  
  05 SIZE-FORMBUFF    COMP VALUE 2000.  
  05 DEVNO            COMP VALUE 1.  
  05 FILLER           COMP VALUE 0.  
  05 FILLER           COMP VALUE 0.  
  05 DATAREC-PACKING  COMP VALUE 2.  
  05 FILLER           COMP VALUE 0.  
  05 FILLER           COMP VALUE 0.  
  05 FILLER           COMP VALUE 0.  
  05 FILLER           COMP VALUE 0.  
  05 FILLER           COMP VALUE 0.
```

* Leaving field situations.

```
01 NUM-LEAVFIELD     COMP VALUE 5.  
01 LEAV-FIELD-FUNC.  
  05 FILLER          COMP VALUE 1002.  
  05 FILLER          COMP VALUE 1003.  
  05 FILLER          COMP VALUE 1004.  
  05 FILLER          COMP VALUE 1005.  
  05 FILLER          COMP VALUE 1000.
```

* Key value for all function keys.

```
01 KEY-VALUE.  
  05 CR              COMP VALUE 13.  
  05 ARROW-RIGHT     COMP VALUE 206.  
  05 ARROW-LEFT      COMP VALUE 194.  
  05 EXIT-KEY        COMP VALUE 163.  
  05 HELP-KEY        COMP VALUE 191.  
  05 F1              COMP VALUE 132.  
  05 F2              COMP VALUE 140.
```

* Function values.

```
01 NUM-KEYS          COMP VALUE 7.  
01 KEY-FUNC.  
  05 CR              COMP VALUE 1000.  
  05 ARROW-RIGHT     COMP VALUE 1000.  
  05 ARROW-LEFT      COMP VALUE 1001.  
  05 EXIT-KEY        COMP VALUE 1.  
  05 HELP-KEY        COMP VALUE 2002.  
  05 F1              COMP VALUE 1000.  
  05 F2              COMP VALUE 1000.
```

* Edit status record.

```
01 NUM-EDSTA         COMP VALUE 6.  
01 EDSTA-REC.  
  05 TERM-CODE       COMP.  
  05 TERM-CHAR       COMP.  
  05 FIELDS-EDITED  COMP.
```

```

05 FIELDS-IN-EDSET      COMP.
05 FIELD-NAME           PIC X(8).
05 OCCURRENCE-NUMBER    COMP.

```

* Message handling variables

```

01 MSG-COUNT            COMP.
01 MSG-MAX              COMP VALUE 30.
01 MSGLINE.
    05 MSG-INDICATOR    PIC X.
    05 MSG-NAME         PIC X(8).
    05 FILLER           PIC X(70).
    05 MSG-END          PIC X.

01 MSGB.
    03 MSGBUFFER        OCCURS 30 TIMES INDEXED BY MSG-I.
        05 MSGB-NAME    PIC X(8).
        05 MSGB-TEXT    PIC X(72).

```

* Form info array.

```

01 FRMARR.
    05 FRMIN-ARR        COMP OCCURS 6 TIMES.
01 NUM-FRMIN            COMP VALUE 6.

01 FORMNAME             PIC X(9).

```

PROCEDURE DIVISION.

* Initiate FOCUS.

```

CALL "FCINITE" USING INITARR FORMBUFF ISTATUS.
MOVE "FCINITE" TO ERRLAB. PERFORM ERRCHECK.

```

* Define edit status buffer.

```

CALL "FCDESB" USING EDSTA-REC NUM-EDSTA ISTATUS.
MOVE "FCDESB" TO ERRLAB. PERFORM ERRCHECK.

```

* Define function keys.

```

CALL "FCCHRBR" USING NUM-KEYS KEY-VALUE KEY-FUNC ISTATUS.
MOVE "FCCHRBR" TO ERRLAB. PERFORM ERRCHECK.

```

* Define leaving field functions.

```

CALL "FCFLDBR" USING NUM-LEAVFIELD LEAV-FIELD-FUNC ISTATUS.
MOVE "FCFLDBR" TO ERRLAB. PERFORM ERRCHECK.

```

* Initiate the message system. The messages are found on the
* form file "TEST", on the forms MSG1 and MSG2.

```

CALL "FCDECFF" USING "TEST" ISTATUS.
MOVE "FCDECFF" TO ERRLAB. PERFORM ERRCHECK.

```



```
MOVE 0 TO MSG-COUNT.  
MOVE "MSG1'" TO FORMNAME.  
PERFORM INIT-MSG.  
MOVE "MSG2'" TO FORMNAME.  
PERFORM INIT-MSG.
```

* Examples on how to use the message system

* Write the message corresponding to the message key MSG3

```
MOVE "^MSG3" TO MSGLINE.  
PERFORM WRITE-MSG.
```

* Write the text 'This was the last example' to the screen.

```
MOVE "This was the last example" TO MSGLINE.  
PERFORM WRITE-MSG.
```

```
STOP RUN.
```

```
*  
* ERRCHECK  
* =====  
* If ISTATUS not equal zero, write an error message to  
* the screen and stop.
```

```
ERRCHECK.  
IF ISTATUS NOT = 0  
  DISPLAY (5, 15) "Error from "  
  DISPLAY (5, 26) ERRLAB  
  DISPLAY (6, 15) "Error code "  
  DISPLAY (6, 26) ISTATUS  
  STOP RUN.
```

```
*  
* INIT-MSG  
* =====  
* This module reads messages from the form FORMNAME on  
* current form file and places the messages in MSGBUFFER.  
* "FCFRMIN" is called to get number of lines in the form.  
* MSG-COUNT = number of messages read.
```

```
INIT-MSG.  
CALL "FCDECFN" USING FORMNAME 0 ISTATUS.  
MOVE "FCDECFN" TO ERRLAB. PERFORM ERRCHECK.  
CALL "FCFRMIN" USING FRMARR NUM-FRMIN ISTATUS.  
MOVE "FCFRMIN" TO ERRLAB. PERFORM ERRCHECK.  
DO FOR I FROM 1 BY 1 TO FRMIN-ARR(3)  
  MOVE SPACES TO MSGLINE.  
  CALL "FCGLINE" USING I I DATARECORD MSGLINE ISTATUS.  
  MOVE "FCGLINE" TO ERRLAB. PERFORM ERRCHECK.  
  ADD 1 TO MSG-COUNT.  
  IF MSG-MAX < MSG-COUNT  
    MOVE "MSGCOUNT" TO ERRLAB
```

```
        MOVE -1 TO ISTATUS
        PERFORM ERRCHECK.
    MOVE MSG-COUNT TO MSG-I.
    MOVE MSGLINE TO MSGBUFFER(MSG-I).
END-DO.
```

```
*           WRITE-MSG
*           =====
* This module checks the first character in MSGLINE.
* If it is a "^", the following 8 characters is used as
* a key to find a message in MSGBUFFER. If found, the
* message from MSGBUFFER is written to the screen,
* else MSGLINE is written to the screen.
```

```
WRITE-MSG.
    IF MSG-INDICATOR = "^"
        DO FOR MSG-I FROM 1 BY 1 TO MSG-COUNT
            IF MSGB-NAME(MSG-I) = MSG-NAME
                MOVE MSGB-TEXT(MSG-I) TO MSGLINE
                EXIT.
        END-DO.
    MOVE "" TO MSG-END
    CALL "FCZMSGE" USING MSGLINE ISTATUS
    MOVE "FCZMSGE" TO ERRLAB. PERFORM ERRCHECK.

EXIT.
```


A P P E N D I X F

PLANC Interface to FOCUS-LIB

FCBELL Bell.
ROUTINE STANDARD VOID,VOID &
(INTEGER WRITE) : FCBELL

FCCHRB Define Function Keys.
ROUTINE STANDARD VOID,VOID &
(INTEGER,INTEGER ARRAY,INTEGER ARRAY,INTEGER WRITE) : FCCHRB

FCCLFDS Clear Fields.
ROUTINE STANDARD VOID,VOID &
(INTEGER,INTEGER WRITE) : FCCLFDS

FCCLFO Remove Form From Screen.
ROUTINE STANDARD VOID,VOID(INTEGER WRITE) : FCCLFO

FCCLMR Clear Must Read.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER,INTEGER WRITE) : FCCLMR

FCCLOSE Close File.
ROUTINE STANDARD VOID,VOID &
(INTEGER,INTEGER WRITE) : FCCLOSE

FCCLREC Clear Data Record.
ROUTINE STANDARD VOID,VOID &
(BYTES WRITE,INTEGER WRITE) : FCCLREC

FCCLSCR Clear Rectangular Area on Screen.
ROUTINE STANDARD VOID,VOID &
(INTEGER,INTEGER,INTEGER,INTEGER,INTEGER WRITE) : FCCLSCR

FCCLSUB Clear Subrecord.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER,BYTES WRITE,INTEGER WRITE) : FCCLSUB

FCDECFF Declare Form File.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER WRITE) : FCDECFF

FCDECFN Declare Form Name.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER WRITE) : FCDECFN

FCDECRC Declare Record.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER,INTEGER WRITE) : FCDECRC

FCDESB Define Edit Status Buffer.
 TYPE EDITSTATUS = RECORD PACKED
 integer : termcode
 integer : termchar
 integer : fdsedited
 integer : fdsineset
 bytes : finame(0:7)
 integer : occno
 integer : pos
 ENDRECORD
 ROUTINE STANDARD VOID, VOID &
 (EDITSTATUS, INTEGER, INTEGER WRITE) : FCDESB

FCEBUF Empty Output Buffer
 ROUTINE STANDARD VOID, VOID(INTEGER WRITE) : FCEBUF

FCEDSTA Get Edit Status.
 ROUTINE STANDARD VOID, VOID &
 (INTEGER WRITE, INTEGER ARRAY WRITE, &
 INTEGER WRITE, INTEGER WRITE) : FCEDSTA

FCEFLD Edit Field.
 ROUTINE STANDARD VOID, VOID &
 (BYTES, INTEGER, BYTES READ WRITE, INTEGER WRITE, &
 INTEGER, INTEGER WRITE) : FCEFLD

FCEREC Edit Record.
 ROUTINE STANDARD VOID, VOID &
 (BYTES READ WRITE, INTEGER, INTEGER WRITE) : FCEREC

FCESFLD Set Edit Start Field.
 ROUTINE STANDARD VOID, VOID &
 (BYTES, INTEGER, INTEGER WRITE) : FCESFLD

FCESFNC Define Edit Start Function.
 ROUTINE STANDARD VOID, VOID &
 (INTEGER, INTEGER, INTEGER, INTEGER WRITE) : FCESFNC

FCESUB Edit Subrecord.
 ROUTINE STANDARD VOID, VOID &
 (BYTES, INTEGER, INTEGER, BYTES READ WRITE, &
 INTEGER, INTEGER WRITE) : FCESUB)

FCFLDBR Define Leaving Field Functions.
 ROUTINE STANDARD VOID, VOID &
 (INTEGER, INTEGER ARRAY, INTEGER WRITE) : FCFLDBR

FCFLDIN Get Field Info.
 ROUTINE STANDARD VOID, VOID &
 (BYTES, INTEGER, INTEGER ARRAY WRITE, &
 INTEGER, INTEGER WRITE) : FCFLDIN

FCFLDNA Get Field Names.
 ROUTINE STANDARD VOID, VOID(INTEGER, INTEGER, &
 BYTES WRITE, INTEGER ARRAY WRITE, INTEGER WRITE) : FCFLDNA

FCFLDST Get Field Status.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER ARRAY WRITE, &
INTEGER,INTEGER WRITE) : FCFLDST

FCFRMBR Define Leaving Form Functions.
ROUTINE STANDARD VOID,VOID &
(INTEGER,INTEGER ARRAY,INTEGER WRITE) : FCFRMBR

FCFRMIN Get Form Information.
ROUTINE STANDARD VOID,VOID &
(INTEGER ARRAY,INTEGER,INTEGER WRITE) : FCFRMIN

FCGLINE Get Form Line.
ROUTINE STANDARD VOID,VOID(INTEGER,INTEGER,BYTES, &
BYTES WRITE,INTEGER WRITE) : FCGLINE

FCGMSGE Get Message.
ROUTINE STANDARD VOID,VOID &
(BYTES,BYTES WRITE,INTEGER WRITE) : FCGMSGE

FCGSUB Get Data Elements From a Data Record.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER,BYTES, &
BYTES WRITE,INTEGER WRITE) : FCGSUB

FCINITE Initiate FOCUS.
ROUTINE STANDARD VOID,VOID &
(INTEGER ARRAY,INTEGER ARRAY,INTEGER WRITE) : FCINITE

FCNXFLD Define Next Field to be Edited.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER,INTEGER WRITE) : FCNXFLD

FCOPEN Open File.
ROUTINE STANDARD VOID,VOID &
(BYTES,BYTES,INTEGER WRITE,INTEGER WRITE) : FCOPEN

FCPCUR Position Cursor.
ROUTINE STANDARD VOID,VOID &
(INTEGER,INTEGER,INTEGER WRITE) : FCPCUR

FCPOSFO Position Form on Screen.
ROUTINE STANDARD VOID,VOID &
(INTEGER,INTEGER,INTEGER WRITE) : FCPOSFO

FCPRDOC Print Document to File.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER WRITE) : FCPRDOC

FCPSUB Put Data Elements into a Data Record.
ROUTINE STANDARD VOID,VOID &
(BYTES,INTEGER,INTEGER,BYTES WRITE, &
BYTES,INTEGER WRITE) : FCPSUB

FCRCHR Read Character.
ROUTINE STANDARD VOID, VOID &
(INTEGER WRITE, INTEGER WRITE) : FCRCHR

FCRDFO Redisplay Forms.
ROUTINE STANDARD VOID, VOID &
(INTEGER, BYTES, INTEGER WRITE) : FCRDFO

FCRTXT Read Text.
ROUTINE STANDARD VOID, VOID &
(INTEGER, INTEGER, BYTES WRITE, &
INTEGER READ WRITE, INTEGER WRITE) : FCRTXT

FCSAVFO Save Form.
ROUTINE STANDARD VOID, VOID &
(INTEGER, BYTES, INTEGER WRITE) : FCSAVFO

FCSCRIN Get Screen Info.
ROUTINE STANDARD VOID, VOID &
(INTEGER ARRAY WRITE, INTEGER, INTEGER WRITE) : FCSCRIN

FCSETAT Set Attributes.
ROUTINE STANDARD VOID, VOID &
(INTEGER, BYTES, INTEGER, INTEGER, INTEGER WRITE) : FCSETAT

FCSETMR Set Must Read.
ROUTINE STANDARD VOID, VOID &
(BYTES, INTEGER, INTEGER, INTEGER WRITE) : FCSETMR

FCWDOTS Write Dots.
ROUTINE STANDARD VOID, VOID &
(BYTES, INTEGER, INTEGER, INTEGER WRITE) : FCWDOTS

FCWFLD Write Field.
ROUTINE STANDARD VOID, VOID &
(BYTES, INTEGER, BYTES, INTEGER, INTEGER WRITE) : FCWFLD

FCWLTXT Write Leading Texts.
ROUTINE STANDARD VOID, VOID &
(INTEGER, INTEGER WRITE) : FCWLTXT

FCWREC Write Record.
ROUTINE STANDARD VOID, VOID &
(BYTES, INTEGER WRITE) : FCWREC

FCWSUB Write Subrecord.
ROUTINE STANDARD VOID, VOID &
(BYTES, INTEGER, INTEGER, BYTES, INTEGER WRITE) : FCWSUB

FCWTXT Write Text.
ROUTINE STANDARD VOID, VOID &
(INTEGER, INTEGER, BYTES, INTEGER, INTEGER WRITE) : FCWTXT

FCZMSGE Send Message.
ROUTINE STANDARD VOID, VOID &
(BYTES, INTEGER WRITE) : FCZMSGE

Index

Aborting commands	19.
ACD	37.
Alphabetic range	26.
Alternative character set	35, 74, 130.
Attributes, display	33.
BCD	37.
Blank when zero	38.
Break functions	77.
Buffer, edit status	87.
CANCEL	81, 113.
key	19.
Character set	25.
Characters, legal	37.
Character set, alternative	35, 74, 130.
Checking routines	38.
CLEAR-DEFINITION	22.
Close file	74.
Command	
summary	17.
types	19.
Commands	17.
Commands, aborting	19.
Control read	66.
Copy	81, 113.
Copying fields	43.
CREATE-FILE	20.
Data element	51.
Data record	51.
field set	51.
Decimal point	25.
Declare	
form file	58.
form name	58.
record	59.
Default value	38.
Define next field	62.
Defining fields	36.
DELETE-FORM	21.
Deleting fields	43.
Direct commands	19.
Display attributes	33, 60, 130.
Document, print	74.
Dummy routines	139, 140.
Edit	
mode	65.
one field	68.
record	67.
status	89.
subrecord	67.
Editing	
a field	109.
text	31.

Edit set	51.
Edit status buffer	87.
ENLARGE-FILE	20.
Environment	24.
source	20.
ENVIRONMENT command	26.
Error codes	125.
EXIT	23.
Expand text	25.
FCBELL	106.
FCCHRBR	84.
FCCLFDS	98.
FCCLFO	95.
FCCLMR	62.
FCCLOSE	74.
FCCLREC	96.
FCCLSCR	105.
FCCLSUB	97.
FCDECFE	58.
FCDECFN	58.
FCDECRC	59.
FCDESB	87.
FCEBUF	106.
FCEDSTA	89.
FCEFLD	68.
FCEREC	67.
FCESFLD	63.
FCESFNC	63.
FCESUB	67.
FCFLDBR	85.
FCFLDNA	92.
FCFLDST	92.
FCFRMBR	86.
FCFRMIN	90.
FCGLINE	100.
FCGSUB	99.
FCINITE	55.
FCMSGE	76.
FCNXFLD	62.
FCOPEN	73.
FCPCUR	105.
FCPOSFO	64.
FCPRDOC	74.
FCPSUB	100.
FCRCHR	106.
FCRDFO	102.
FCRTXT	103.
FCSAVFO	94.
FCSCRIN	90.
FCSETAT	60.
FCSETMR	61.
FCSRIN	91.

FCUCONT	38, 121.
FCUFUNC	82.
FCWDOTS	98.
FCWFLD	72.
FCWLTXT	101.
FCWREC	71.
FCWSUB	71.
FCWTXT	104.
FCZMSGE	75.
Field	51.
definition	36.
format	36, 39.
groups	27.
indicator	26.
information	91.
key	36.
name	36, 51, 92.
status	92.
types	39.
Field,	
edit one	68.
write one	72.
Fields	
numeric	39.
string	39.
text	39.
Fields,	
marking	43.
modifying	43.
moving, copying or deleting	43.
Fields per line, maximum number of	7.
Field group	52.
name	52.
Field list	52.
FOCUS-COMPIL	117.
FOCUS-RTS	22, 49.
Form	51.
extent	44.
information	90.
Format, field	36.
Forms, loaded	117.
Form editing, terminating	112.
Form file commands	20.
Form file, declare	58.
Form name, declare	58.
FUNC @	23.
Get Message	76.
GET-ENVIRONMENT	25.
GET-KEY-VALUE	23.
Groups	27.
HELP	7, 80, 113.
at runtime	37.

Help form	37.
HOME key	9.
Illegal values	38.
Information calls	87.
Initial value	38.
Integer	37.
Internal representation	42.
Introduction	3, 7.
Justification	38.
Key value	53.
Leading text	51.
Legal	
characters	37.
values	38.
List of commands	17.
LIST-FILES	20.
LIST-FORMS	22.
Loaded forms	117.
Loading FOCUS applications	139.
MAKE-UP-TO-DATE-FORMS	26.
Marked	
fields	31, 43.
text	31.
Maximum number of fields per line	7.
Message Calls	75.
Mini FOCUS	129.
Modifying fields	43.
Moving fields	43.
Must read,	
clear	62.
set	61.
Navigating in a form	78, 111.
Navigation	53.
Number group separator	25.
Numeric	
E-format	39.
fields	39.
Occurrence range	28, 38.
Occurrence number	51.
Occurrence numbers	44.
in group definitions	27.
Occurrence part	51.
One field,	
edit	68.
write	72.
Open file	73.
Packed decimal	37.
Parameter types	54.
Password	66.
Precision	43.
Print document on file	74.
Questionnaires	19.

Range	43.
READ-FORM	21.
Record,	
declare	59.
edit	67.
write	71.
Redisplay	81, 113.
screen	23.
Screen Information	90.
Screen, redisplay	23.
Send message	75.
Separator, number group	25.
Set display	60.
SINTRAN commands	28.
Status	49.
calls	87.
Storage type	37, 42.
String fields	39.
Subrecord,	
edit	67.
write	71.
Summary, command	17.
System	
messages	25.
routines	38.
Terminal types	129.
Terminal operator interface	77, 109.
Terminating form editing	112.
Termination code	78, 88.
Text	
editing	31.
fields	39.
Text, marked	31.
Unpacked decimal	37.
User defined control	121.
Utility commands	22.
Values,	
illegal	38.
legal	38.
Write	
one field	72.
record	71.
subrecord	71.
WRITE-FORM	19, 21.

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: FOCUS Screen Handling System Reference Manual

Manual number: ND-60.137.5 EN

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 ? _____

NOTE!

This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:

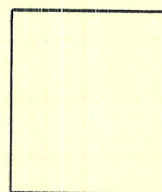
Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Norsk Data's answer will be found on reverse side



[illegible]

Date



Documentation Department
P.O. Box 25, Bogerud
0621 Oslo6, Norway

Systems that put people first

NORSK DATA A.S OLAF HELSETHS VEI 5 P.O. BOX 25 BOGERUD 0621 OSLO 6 NORWAY
TEL.: 02 - 62 60 00 - TELEX: 18284 NDN