

WebFOCUS

Designing Screens With FIDEL
Version 4 Release 3.1

Designing Screens With FIDEL

1	Designing Screens With FIDEL	1-1
	Getting Started Using FIDEL With Modify	1-2
	Describing the CRT Screen	1-4
	Specifying Elements of the CRTFORM.....	1-4
	Defining a Field.....	1-5
	Using Spot Markers for Text and Field Positioning.....	1-6
	Specifying Lowercase Entry: UPPER/LOWER.....	1-8
	Data Entry, Display, and Turnaround Fields.....	1-9
	Controlling the Use of PF Keys.....	1-13
	Resetting PF Key Controls.....	1-14
	Setting PF Key Fields for Branching Purposes	1-14
	Using Labeled Fields.....	1-16
	Specifying Cursor Position.....	1-17
	Determining Current Cursor Position for Branching Purposes	1-18
	Using FIDEL in MODIFY	1-21
	Conditional and Non-Conditional Fields.....	1-21
	Using FIXFORM and FIDEL in a Single MODIFY	1-26
	Default CRTFORM Processing Within the Same MODIFY Request.....	1-28
	Generating Automatic CRTFORMs.....	1-30
	Using Multiple CRTFORMs: LINE.....	1-34
	CRTFORMs and Case Logic	1-41
	Specifying Groups of Fields.....	1-43
	Using REPEAT to Display Multiple Records	1-45
	Using Groups of Fields With Case Logic.....	1-49
	Handling Errors	1-53
	Handling Format Errors.....	1-53
	VALIDATE and CRTFORM Display Logic	1-54
	Handling Errors With Repeating Groups	1-55
	Rejecting NOMATCH or Duplicate Data	1-57
	Logging Transactions	1-58
	Additional Screen Control Options	1-58
	Clearing the Screen: CLEAR/NOCLEAR	1-59
	Index	I-1

Cactus, EDA, FIDEL, FOCCALC, FOCUS, FOCUS Fusion, Information Builders, the Information Builders logo, SmartMode, SNAPpack, TableTalk, and Web390 are registered trademarks and Parlay, SiteAnalyzer, SmartMart, and WebFOCUS are trademarks of Information Builders, Inc.

Acrobat and Adobe are registered trademarks of Adobe Systems Incorporated.

NOMAD is a registered trademark of Aonix.

UniVerse is a registered trademark of Ardent Software, Inc.

IRMA is a trademark of Attachmate Corporation.

Baan is a registered trademark of Baan Company N.V.

SUPRA and TOTAL are registered trademarks of Cincom Systems, Inc.

Impromptu is a registered trademark of Cognos.

Alpha, DEC, DECnet, NonStop, and VAX are registered trademarks and Tru64, OpenVMS, and VMS are trademarks of Compaq Computer Corporation.

CA-ACF2, CA-Datcom, CA-IDMS, CA-Top Secret, and Ingres are registered trademarks of Computer Associates International, Inc.

MODEL 204 and M204 are registered trademarks of Computer Corporation of America.

Paradox is a registered trademark of Corel Corporation.

StorHouse is a registered trademark of FileTek, Inc.

HP MPE/iX is a registered trademark of Hewlett Packard Corporation.

Informix is a registered trademark of Informix Software, Inc.

Intel is a registered trademark of Intel Corporation.

ACF/VTAM, AIX, AS/400, CICS, DB2, DRDA, Distributed Relational Database Architecture, IBM, MQSeries, MVS, OS/2, OS/400,

RACF, RS/6000, S/390, VM/ESA, and VTAM are registered trademarks and DB2/2, HiperSpace, IMS, MVS/ESA, QMF, SQL/DS, VM/XA and WebSphere are trademarks of International Business Machines Corporation.

INTERSOLVE and Q+E are registered trademarks of INTERSOLVE.

Orbit is a registered trademark of Iona Technologies Inc.

Approach and DataLens are registered trademarks of Lotus Development Corporation.

ObjectView is a trademark of Matesys Corporation.

ActiveX, Microsoft, MS-DOS, PowerPoint, Visual Basic, Visual C++, Visual FoxPro, FrontPage, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

Teradata is a registered trademark of NCR International, Inc.

Netscape, Netscape FastTrack Server, and Netscape Navigator are registered trademarks of Netscape Communications Corporation.

NetWare and Novell are registered trademarks of Novell, Inc.

CORBA is a trademark of Object Management Group, Inc.

Oracle is a registered trademark and Rdb is a trademark of Oracle Corporation.

PeopleSoft is a registered trademark of PeopleSoft, Inc.

INFOAccess is a trademark of Pioneer Systems, Inc.

Progress is a registered trademark of Progress Software Corporation.

Red Brick Warehouse is a trademark of Red Brick Systems.

R/3 and SAP are registered trademarks of SAP AG.

Silverstream is a trademark of Silverstream Software.

ADABAS is a registered trademark of Software A.G.

CONNECT:Direct is a trademark of Sterling Commerce.

Java, JavaScript, NetDynamics, Solaris, and SunOS are trademarks of Sun Microsystems, Inc.

PowerBuilder and Sybase are registered trademarks and SQL Server is a trademark of Sybase, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

Three D Graphics is a trademark and Perspective is a registered trademark of Three D Graphics, Inc. Portions of WebFOCUS Desktop Graph Editor documentation are adapted from Perspective® for Java documentation.

Due to the nature of this material, this document refers to numerous hardware and software products by their trade names. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies. It is not this publisher's intent to use any of these names generically. The reader is therefore cautioned to investigate all claimed trademark rights before using any of these names other than to refer to the product described.

Copyright © 2000, by Information Builders, Inc. All rights reserved. This manual, or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Printed in the U.S.A.

CHAPTER 1

Designing Screens With FIDEL

Topics:

- Getting Started Using FIDEL With Modify
- Describing the CRT Screen
- Using FIDEL in MODIFY
- Handling Errors

The FOCUS Interactive Data Entry Language, FIDEL, enables you to design full-screen forms for data entry and application development. FIDEL is used with MODIFY to build data maintenance and inquiry screens.

Note:

- Dialogue Manager (-CRTFORM) is not supported in this release. Additionally, MODIFY CRTFORM only displays output in HTML format (Desktop Viewer) and therefore it does not support field attributes such as color and highlighting.
- The screens in this chapter show the text output only. The appearance of the screens will be different in HTML format (Desktop Viewer).

Getting Started Using FIDEL With Modify

When you use FIDEL with MODIFY, you are setting up full-screen forms for the maintenance of data source fields. Most MODIFY features such as conditional and non-conditional fields, automatic application generation, Case Logic, multiple record processing, error handling, validation tests, logging transactions, and typing messages to the terminal work with FIDEL.

With MODIFY you also have access to additional screen control options such as clearing the screen, specifying and changing the size of the screen, and designating the particular line on which the form starts.

Example

Using FIDEL With MODIFY

The following example of a simple MODIFY CRTFORM illustrates the use of FIDEL:

```
MODIFY FILE EMPLOYEE
1. CRTFORM
2.   "EMPLOYEE UPDATE"
3.   "EMPLOYEE ID #:   <EMP_ID           LAST NAME:   <LAST_NAME"
4.   "DEPARTMENT:     <DEPARTMENT       SALARY:     <CURR_SAL"

5. MATCH EMP_ID
   ON NOMATCH REJECT
   ON MATCH UPDATE LAST_NAME DEPARTMENT CURR_SAL
6. DATA VIA FIDEL
   END
```

This request sets up a form to update the last name, department, and current salary. The procedure processes as follows:

1. CRTFORM generates the visual form and invokes FIDEL. The form begins on line one of the screen unless specified otherwise with the LINE option. See *Using Multiple CRTFORMs: LINE* on page 1-34 for details on the LINE option.
2. Each line on the screen begins and ends with double quotation marks. This is a line of text that serves as a title. Note the close correspondence to the syntax used to create headings in a TABLE request.
3. The second screen line specifies two data fields: EMP_ID and LAST_NAME. A data entry field is indicated by a left caret, followed by the field name or alias from the Master File. The text, EMPLOYEE ID #: and LAST NAME: identifies each field on the screen. This informs the operator where to enter the data.

4. This is the last line within double quotation marks. It signals the end of the CRTFORM. In this case it identifies and defines two more data fields: DEPARTMENT and CURR_SAL. When you execute the MODIFY request, the form instantly displays on the screen:

```
EMPLOYEE UPDATE
EMPLOYEE ID #:          LAST NAME:
DEPARTMENT:            SALARY:
```

The number of characters allotted for each data entry field on the screen defaults to the display format for that particular field in the Master File. You can optionally specify a format for screen display that is shorter than the default.

The operator can now fill in the data entry areas with the appropriate information.

5. The request continues with MODIFY MATCH logic.
6. The last line tells FOCUS where the incoming data is from. In WebFOCUS Desktop, you must use DATA VIA FIDEL to indicate the incoming data is from the Desktop Viewer.

Reference

Usage Notes for Using FIDEL Screens: Operating Conventions

The following procedures apply for filling in *all* FIDEL screens:

- To move from field to field, press the Tab key. You can also move the cursor around the screen using the arrow keys.
- When filling in values on the screen, you may use any of the keys on the keyboard.
- To scroll forward or backward through a long CRTFORM (from screen to screen), use the scroll bar. Note that in previous versions, scrolling was performed by pressing the PF8 or PF7 key.
- To transmit the screen, press the Enter key.
- If you make an error, the transaction may not be transmitted and an error message may display at the bottom of the screen. You can correct the error and retransmit the screen.
- To signal the end of data entry, press the PF3 key or type END in an unprotected area. In MODIFY, this terminates the request.
- To return to the first screen without transmitting the current screen, press the PF2 key or the key set to QUIT.
- The Reset key clears all entries.

Note: The PF key settings referred to here are the default settings. Any PF key can be redefined using the SET statement.

Describing the CRT Screen

The MODIFY statement CRTFORM followed by the screen layout, generates a form. Within one MODIFY procedure you can use an unlimited number of screen lines (within memory constraints). Each screen line can contain a maximum of 78 characters of text and data.

In MODIFY, you can use up to 255 CRTFORM statements in a procedure and a maximum of 248 lines are permitted for each CRTFORM.

Specifying Elements of the CRTFORM

To create the visual form, you enter the screen lines one after the other within double quotation marks. For each screen line, you can specify various screen elements such as descriptive text and fields. A left caret (<) followed by the name of the field generates the position where data is to be entered onto the screen.

You may need to use two FOCEXEC lines to describe one physical CRTFORM line. Simply omit the double quotation marks (“) at the end of the first line and omit them at the beginning of the next line as well. Everything between the set of double quotation marks will read as one screen line on the CRTFORM.

Syntax

How to Invoke FIDEL Using CRTFORM

The following is a summary of the complete syntax for generating a CRTFORM in MODIFY. The individual options and screen elements are described in detail in specific sections later in the chapter.

```
CRTFORM [option option...]  
"screen element [screen element...]"
```

where:

CRTFORM

Automatically invokes FIDEL and sets up the visual form. Subsequent lines describe the screen.

option option...

Refers to screen control options.

"screen element..."

Can be user-defined text, fields, or spot markers. Spot markers define the next place on the screen where a screen element will display. Both spot markers and fields are preceded by a left caret and optionally closed by a right caret.

Note:

- You can use the asterisk (*) with CRTFORM in FIDEL to generate a CRTFORM containing all of the data source's fields automatically (that is, without specifying individual fields). See *Generating Automatic CRTFORMs* on page 1-30 for information on CRTFORM *, its syntax, and variations.
- Do not begin any field used in a CRTFORM or FIXFORM statement with Xn, where n is any numeric value. This applies to fields in the Master File and computed fields.

Defining a Field

Labels, prefixes, attributes, and formats are parts of the definition of a particular field.

Note: Fields with a text (TX) format cannot be used in CRTFORM. However, they can be entered interactively using TED.

Syntax

How to Define a Field

```
<[:label.] [prefix.] [attribute.] field[/length] [>]
```

where:

label

Is a user-defined label of up to 12 characters associated with a field. It may not contain embedded blanks. For more information on user-defined labels, see *Using Labeled Fields* on page 1-16.

prefix

Designate a display or turnaround field (D. or T.), respectively. See *Data Entry, Display, and Turnaround Fields* on page 1-9 for more information.

attribute

Is the abbreviation or full name of a screen attribute.

field

Is the name of the field or variable being defined.

length

Is the length of the field as it appears on the screen. In MODIFY, you need to define a length, only if you want the screen length to be different from the format length that is defined in the Master File or COMPUTE command.

Note: When you use the abbreviations for attributes, you do not need to use the dot separator between attributes or between a prefix and an attribute. See *Data Entry, Display, and Turnaround Fields* on page 1-9 for more information on prefixes and *Using Labeled Fields* on page 1-16 for more information on labels.

Using Spot Markers for Text and Field Positioning

Because the lengths of fields vary, text does not automatically align uniformly on the screen. Spot markers are available to help you position both text and fields. A spot marker is essential to eliminate trailing blanks at the end of the first line, if your screen line description takes up two FOCEXEC lines.

Reference

How to Use Spot Markers to Position Text and Fields

Marker	Example	Usage
<n or <n>	<50	Positions the next character in column 50.
<+n or <+n>	<+4	Positions the next character four columns from the last non-blank character.
<-n or <-n>	<-1	Positions the next character one column to the left of the last character. This marker's function is to suppress or write over the attribute byte at the beginning and the end of a field.
</n or </n>	</2	Positions the next character at the beginning of the line, that is two lines from the last (skips two lines). Note: The last line is blank and is created when a double quotation mark (") is encountered.
<0X or <0X>	<0X	Positions the next character immediately to the right of the last character (skip zero columns). This is used to help position data on a FIDEL screen when a single screen line is coded as two lines in a FOCEXEC. No spaces are inserted between the spot marker and the start of a continuation line.

Note: You can optionally use the right caret >. This is useful when the next character in the line is a left caret. It also enhances readability.

Example

Using Spot Markers for Positioning

Suppose you want the various input data fields arranged across the screen in vertical sections, left justified, and in horizontal segments marked off with lines. In the following example, spot markers are used to create the desired screen.

```

MODIFY FILE EMPLOYEE
CRTFORM
    "EMPLOYEE UPDATE"
1.    "</1"
    "-----"
    "EMPLOYEE ID #: <EMP_ID   LAST NAME: <LAST_NAME"
2.    "</1"
3.    "DEPARTMENT: <DEPARTMENT <+3 CURRENT SALARY:<0X>
    <CURR_SAL"
    "-----"
    "BANK: <BANK_NAME"
    "-----"
MATCH EMP_ID
    .
    .
    .
DATA VIA FIDEL
END

```

The spot markers in the example perform the following functions:

- 1/2. </1 skips one blank line.
3. <+3 moves the word CURRENT three spaces to the right of the last letter in the word DEPARTMENT. <0X> skips no spaces. No extra spaces are inserted between this and the next word (<CURR_SAL) on the continuation line. There is, in fact, one space before the field which is an attribute byte that marks the start of a field.

The screen displays as follows:

```

EMPLOYEE UPDATE

-----
EMPLOYEE ID #: LAST NAME:

DEPARTMENT:           CURRENT SALARY:
-----
BANK:
-----

```

Specifying Lowercase Entry: UPPER/LOWER

All entered text is normally translated to uppercase letters. You can override this default and preserve both uppercase and lowercase text by using the lowercase option.

Syntax

How to Use the Lowercase Option

```
[ - ]CRTFORM [ UPPER | LOWER ]
```

where:

UPPER

Translates all characters to uppercase. This is the default.

LOWER

Reads lowercase data from the screen. Once you specify LOWER, every screen thereafter is a lowercase screen until you specify UPPER.

Note: In MODIFY, when you use multiple CRTFORMs on the same screen (using LINE n), you can mix UPPER and LOWER among the forms.

Example

Using the Lowercase Option

The following example shows the syntax of a simple MODIFY CRTFORM using the lowercase option, followed by two screen lines containing various screen elements: text, a spot marker, and a field.

1. CRTFORM LOWER
2. "PLEASE FILL IN THE EMPLOYEE ID # </1"
3. "EMPLOYEE ID #: <EMP_ID"
MATCH EMP_ID
.
.
.

The procedure processes as follows:

1. CRTFORM invokes FIDEL and generates the form. The LOWER case option specifies that what is entered from the terminal in lowercase will remain in lowercase.
2. The first line of the screen contains descriptive text.
</1 is a spot marker which skips one blank line.

3. The last line of the screen contains two screen elements: descriptive text that identifies the field and the data source field EMP_ID. The last line between quotation marks signals the end of the CRTFORM.

The form generated displays as follows:

```
PLEASE FILL IN THE EMPLOYEE ID #  
EMPLOYEE ID #:
```

Data Entry, Display, and Turnaround Fields

There are three types of data or variable fields that can be specified on the CRTFORM: data entry, display, and turnaround.

You can also compute data fields and specify them as entry, display, or turnaround on the CRTFORM. You can convert a turnaround field to a display field dynamically.

In MODIFY, fields can also be designated as conditional or unconditional. We recommend that for data entry, you use conditional fields (left caret only) so that the values in your data source are not replaced by a blank or a zero if you do not enter data for the field. See *Conditional and Non-Conditional Fields* on page 1-21 for a complete explanation of the use of conditional and non-conditional fields in MODIFY.

For most turnaround fields, we recommend that you use non-conditional fields (both carets). A non-conditional turnaround field remains active whether you enter data or not. Because the value in the data source is displayed in the field, that value remains in the data source if you do not change it. Because the field remains active, the values for your VALIDATEs and COMPUTE s are then accurate.

Syntax

How to Specify Different Field Types

Data entry (for data entry only):

`<field[/length] [>]`

where:

field

Is the name of the field. Reserves space on the screen for data entry into that field and does not display the current value of the field.

In MODIFY, if only the left caret is used, data entry is conditional. If both carets are used, the field is non-conditional. For more information on non-conditional fields, see *Conditional and Non-Conditional Fields* on page 1-21.

Display (for information only):

Data is displayed in a protected area and cannot be altered.

`<D.field[/length]`

where:

D.

Is the prefix placed in front of a field, indicating that the data or value is to be displayed. The current value of the field displays on the screen, but in a protected area which cannot be changed.

Note that the right caret is meaningless for display fields.

Turn-around field (for display and change):

Data is displayed in an unprotected area and can be altered.

`<T.field[/length] [>]`

where:

T.

Is the prefix placed in front of a field to indicate that it is a turnaround field. The current value of the field is displayed on the screen. However, the operator may change the value, as it is not in a protected area. There is a limitation of 300 unique T. fields that can be displayed on one CRTFORM.

In MODIFY, if only the left caret is present, the T. field is treated as conditional. If the right caret is used, the field is non-conditional, and the value is treated as present, even if unchanged.

Note: In MODIFY, in order to display data from a data source field or present it for turnaround, a position in the data source must first be established through the use of a MATCH or NEXT statement or value must be assigned in a COMPUTE. A computed field cannot be set and displayed in the TOP case because in the TOP case, data entry is processed prior to computations. For example, one of the phrases

```
ON MATCH CRTFORM
ON NEXT CRTFORM
```

must be used. A position is thus established in the data source, and the values of the fields in existing records are now available for display as protected or unprotected fields.

You can also match on a key field and go to a case in which you display a CRTFORM using display and turnaround fields.

Example

Using Data Entry, Display, and Turnaround Fields

The following example combines two CRTFORMs in a single MODIFY request and shows the use of entry, display, and turnaround fields.

```
MODIFY FILE EMPLOYEE
1. CRTFORM
   "ENTER EMPLOYEE ID#: <EMP_ID>"
   "PRESS ENTER"
   "</2>"
2. MATCH EMP_ID
   ON NOMATCH REJECT
2.   ON MATCH CRTFORM
   " "
   "REVISE DATA FOR SALARY AND DEPARTMENT"
   "ENTER NEW DATA FOR EDUCATION HOURS"
   " "
3.   "EMPLOYEE ID #: <D.EMP_ID>  LAST_NAME: <D.LAST_NAME>"
   " "
4.   "SALARY:           <T.CURR_SAL>"
   "DEPARTMENT:       <T.DEPARTMENT>"
5.   "EDUCATION HOURS:   <ED_HRS>"
   ON MATCH UPDATE CURR_SAL DEPARTMENT ED_HRS
DATA VIA FIDEL
END
```

The procedure matches the employee ID, displays both the ID and the last name, and then displays the current salary and department for turnaround. Education hours is a data entry field.

Note that when the procedure executes, both CRTFORMs are displayed immediately. However, the display and turnaround fields in the second CRTFORM do not display data until the operator fills in the first form and presses Enter. We therefore recommend you use the LINE option.

When a FORMAT ERROR occurs, all data entered up to that point is processed and cannot be changed in the course of your transaction.

The processing is as follows:

1. CRTFORM generates the first form which begins on line 1 (the second CRTFORM is displayed, but without values):

```
ENTER EMPLOYEE ID #:  
PRESS ENTER  
  
REVISE DATA FOR SALARY AND DEPARTMENT  
ENTER NEW DATA FOR EDUCATION HOURS  
  
EMPLOYEE ID #:          LAST NAME:  
SALARY:  
DEPARTMENT:  
EDUCATION HOURS:
```

2. The procedure continues with the MATCH logic. If the ID number that is input matches an ID in the data source, the display and turnaround fields on the second CRTFORM display the data. Assume the operator enters 818692173 and presses Enter.

The following is displayed:

```
ENTER EMPLOYEE ID #: 818692173  
PRESS ENTER  
  
REVISE DATA FOR SALARY AND DEPARTMENT  
ENTER NEW DATA FOR EDUCATION HOURS  
  
EMPLOYEE ID #: 818692173    LAST NAME: CROSS  
SALARY:                27062.00  
DEPARTMENT:            MIS  
EDUCATION HOURS:
```

3. This screen line contains two display fields.
4. The next two screen lines contain turnaround fields.
5. The last line is a data entry field.

Note: To display fields from a unique segment, the ON MATCH CONTINUE TO, ON NEXT, or MATCH WITH-UNIQUES phrase must have been executed.

Computed fields in MODIFY can be displayed in any kind of CRTFORM.

Controlling the Use of PF Keys

Note: Keyboard functionality is not supported in this release. You can use your mouse to click on individual PF keys.

The terminal operator can use certain PF keys to control the execution of a FIDEL application. Normally the following keys are used:

- PF3 and PF15 mean END and terminate execution.
- PF2 means Cancel and cancels the transaction in MODIFY.

Note: All other keys return the value of the PF key when pressed.

Several facilities are available to assist you in controlling various screen operations:

- You can reset PF key functions. You can also set PF keys to branch to particular cases in MODIFY.
- You can set the cursor on a specified position on the screen. See *Specifying Cursor Position* on page 1-17 for more information.
- You can use the cursor position on the screen to perform a branch or action based on a test. For more information, see *Determining Current Cursor Position for Branching Purposes* on page 1-18.

Reference

Default Settings for PF Keys

PF Key	Function
PF01	HX
PF02	CANCEL
PF03	END
PF04	RETURN
PF05	RETURN
PF06	RETURN
PF07	BACKWARD Note: Not functional in HTML format (Desktop Viewer).
PF08	FORWARD Note: Not functional in HTML format (Desktop Viewer).
PF09	RETURN
PF10	RETURN

Resetting PF Key Controls

You can reset PF key functions in FIDEL for CRTFORMs using the FOCUS SET command.

Syntax

How to Reset PF Key Functions Using FOCUS SET Commands

```
SET PFxx = function
```

where:

xx

Is a one or two-digit PF key number.

function

Is one of the following:

END in MODIFY, exits the procedure.

CANCEL in MODIFY, cancels the transaction and returns to the TOP case.

FORWARD pages forward.

BACKWARD pages backward.

RETURN has no specific screen action. Returns the PF key name in the PFKEY field because it is not yet defined. To set the PFKEY field, use COMPUTE in MODIFY.

HELP displays text supplied with the HELPMESSAGE attribute for any field on the MODIFY CRTFORM. Position the cursor on the data entry area of the desired field, and press the PF key you have defined for HELP. If no help message exists for that field, the following message is displayed:

```
NO HELP AVAILABLE FOR THIS FIELD.
```

Note: When changing PF key settings, make sure that at least one key is set to END. If you set a PF key to FORWARD, you should also set one to BACKWARD.

Setting PF Key Fields for Branching Purposes

You can create a menu of processing options. The operator can then indicate a choice by pressing a particular PF key. To assign a specific processing function to a PF key, you must specify a field named PFKEY. Which PF key the operator presses determines the value of the PFKEY field.

You can use the PF keys designated as Return keys, as well as the Enter key. You define a variable called PFKEY (in MODIFY) and then test its value after the CRTFORM is displayed. Which branch takes place depends on which PFKEY the operator presses.

Syntax**How to Set PF Keys for Branching Purposes**

```
COMPUTE
PFKEY/A4=;
```

where:

```
PFKEY/A4
```

Is a four-character field, whose value is determined by which key the operator presses at run time.

Example**Testing PF Keys in MODIFY**

1. COMPUTE


```
PFKEY/A4=;
```
2. CRTFORM


```
"SELECT OPTION"
"INPUT  PRESS PF4"
"UPDATE PRESS PF5"
"DELETE PRESS PF6"
```
3. IF PFKEY EQ 'PF04' GOTO INCASE


```
ELSE IF PFKEY EQ 'PF05' GOTO UPCASE
ELSE IF PFKEY EQ 'PF06' GOTO DELCASE
ELSE GOTO TOP;
```

```

.
.
.
```

The example processes as follows:

1. The COMPUTE statement specifies a four-character field PFKEY.
2. CRTFORM generates the form which supplies the operator with three options:

```
SELECT OPTION
INPUT  PRESS PF4
UPDATE PRESS PF5
DELETE PRESS PF6
```

3. The IF test determines what case to branch to depending on the value of the PFKEY field. For example, if the operator presses PF4, the value for PFKEY is PF04, and the request branches to an input case INCASE.

Using Labeled Fields

You can use labels to identify a specific field on the screen. They are necessary to perform the following functions:

- Dynamically change screen attributes during processing depending on the results of tests.
- Position the cursor on the screen, or read the position of the cursor on the screen, where there is no pre-existing field.

Syntax

How to Use Labels to Identify a Specific Field on Screen

`<:label.field`

where:

label

Is a user-defined label. It starts with a colon (:) and may be up to 66 characters long including the colon. You may not use embedded blanks.

field

Is any field on the CRTFORM. It can be a field created specifically for appending a label.

Reference

Usage Notes for Labeling Fields

- A label cannot occur by itself. It must be used with a field.
- A label must be declared via a COMPUTE, -SET, or -DEFAULTS statement.
- Setting a label to \$ returns its field to the default attribute.

Example

Creating a Label via a COMPUTE

In the following example, the label ONE is set to a format of A6 and is the identifier of the field EMP_ID.

```
COMPUTE
:ONE/A6= '    ';
CRTFORM
"<:ONE.EMP_ID"
```

Specifying Cursor Position

To specify cursor position you simply specify the field where you want the cursor positioned. You may specify the field by its field name or by its label. You can set the cursor at a specific place on the screen by computing or setting the value of the field CURSOR.

Syntax

How to Control Cursor Position in MODIFY

The following syntax is for the field which controls the cursor position in MODIFY.

```
COMPUTE
CURSOR/A66= expression;
```

where:

```
CURSOR/A66
```

Is a 66-character alphanumeric field.

```
expression
```

Is terminated with a semicolon and can be anything, including the full field name, its full alias, a unique truncation of either, or the label itself. This determines the position of the cursor.

Examples

Determining the Cursor Position

```
COMPUTE
CURSOR/A66=IF TESTNAME GT 100 THEN 'EMP_ID'
ELSE 'LAST_NAME';
```

The position of the cursor will be on the field EMP_ID if the value of test name is greater than 100, or it will be on the field LAST_NAME if test name is less than or equal to 100.

Positioning the Cursor Using a Field Label

```
COMPUTE
CURSOR/A66=IF TESTNAME GT 100 THEN ':ONE'
ELSE ':TWO';
```

Note: If the field name is not unique, FIDEL uses the first occurrence of the field name (left to right across each line and then down to the next line) to set or test the cursor position.

In MODIFY, the variable CURSORINDEX can also be used to compute the position of the cursor at a particular record when there are multiple indexed records displayed in a single CRTFORM. A common use of this feature is for placing the cursor on invalid fields after VALIDATE statements.

Syntax

How to Use the Variable CURSORINDEX

COMPUTE
CURSORINDEX/I5=*expression*;

where:

CURSORINDEX/I5

Is a five-digit integer field. Refers to the current value of the subscript being processed from the CRTFORM.

expression

May be any expression, but in most applications will be set equal to REPEATCOUNT.

Determining Current Cursor Position for Branching Purposes

Rather than having the operator type a response, you can create a menu on which you list options. To select an option, the operator moves the cursor to the correct line on the screen and presses the Enter key. FOCUS senses the cursor position and takes action based upon it (such as branching to a particular case or field).

To do this, you must specify a 66 character field that contains the current cursor position, CURSORAT. You may identify a field on the screen by a label or by its field name.

Syntax

How to Define the Field Used to Read the Cursor Position

COMPUTE
CURSORAT/A66=;

where:

CURSORAT/A66

Is the field whose value is determined by the field name or label of the field on which the cursor is positioned when the operator presses Enter.

If the actual cursor position is not on any field, the value of CURSORAT is the nearest preceding field. If there are no preceding fields, the value of CURSORAT is the TOP of the CRTFORM. That is, the value is at the very beginning of the CRTFORM.

Example

Using CURSORAT to Determine Cursor Position

In the following example, field XYZ is a computed field for the purpose of creating a labeled field wherever necessary on the CRTFORM.

```

MODIFY FILE EMPLOYEE
1. COMPUTE
   CURSORAT/A66=;
2. :ADD/A1=;
   :UPP/A1=;
3. XYZ/A1=;
4. CRTFORM
   "POSITION CURSOR NEXT TO OPTION DESIRED"
   "THEN PRESS ENTER"
   " "
   "<:ADD.XYZ ADD RECORDS"
   "<:UPP.XYZ UPDATE RECORDS"
5. IF CURSORAT EQ ':ADD' GOTO ADD ELSE
   IF CURSORAT EQ ':UPP' GOTO UPP ELSE GOTO TOP;

CASE ADD
CRTFORM LINE 1
"THIS CRTFORM ADDS RECORDS"
" "
"EMPLOYEE ID #: <EMP_ID"
"LAST NAME: <LAST_NAME"
"FIRST NAME: <FIRST_NAME"
"HIRE DATE: <HIRE_DATE"
"DEPARTMENT: <DEPARTMENT"
MATCH EMP_ID
  ON MATCH REJECT
  ON NOMATCH INCLUDE
ENDCASE

CASE UPP
CRTFORM LINE 1
"THIS CRTFORM UPDATES RECORDS"
" "
"EMPLOYEE ID #: <EMP_ID"
"DEPARTMENT: <DEPARTMENT"
"JOB CODE: <CURR_JOBCODE"
"SALARY: <CURR_SAL"
MATCH EMP_ID
  ON NOMATCH REJECT
  ON MATCH UPDATE DEPARTMENT CURR_JOBCODE CURR_SAL
ENDCASE
DATA VIA FIDEL
END

```

This example processes as follows:

1. The COMPUTE establishes the field CURSORAT.
2. The second and third COMPUTEs declare the labels :ADD and :UPP.
3. The fourth COMPUTE establishes a field XYZ for the purpose of using labels.
4. CRTFORM generates the following visual form beginning on the first line of the screen:

```
POSITION CURSOR NEXT TO OPTION DESIRED  
THEN PRESS ENTER  
  
ADD RECORDS  
UPDATE RECORDS
```

5. An IF phrase tests the value of the field CURSORAT. If the operator places the cursor next to ADD RECORDS, the value of CURSORAT is :ADD and a branch to Case ADD will be performed. If the operator places the cursor next to UPDATE RECORDS, the value of CURSORAT is :UPP and Case UPP will be performed.

Using FIDEL in MODIFY

The following standard MODIFY functions and concepts work with FIDEL in the building of CRTFORMs:

- Conditional and non-conditional field specification. See *Conditional and Non-Conditional Fields* on page 1-21 for details.
- The FIXFORM statement which can be used before the first CRTFORM. This enables you to mix data sources. For more information, see *Using FIXFORM and FIDEL in a Single MODIFY* on page 1-26.
- Automatic application generation which enables you to use several simple statements to generate automatic data management procedures and CRTFORMs. See *Generating Automatic CRTFORMs* on page 1-30 for details.
- Multiple CRTFORMs for different processing options. The additional FIDEL facility of the LINE option helps you organize the use of multiple CRTFORMs. See *Using Multiple CRTFORMs: LINE* on page 1-34 for more information.
- Case Logic which enables you to divide the processing into logical subdivisions for particular sets of circumstances. For more information, see *CRTFORMs and Case Logic* on page 1-41.
- Groups of fields. See *Specifying Groups of Fields* on page 1-43 for details.
- VALIDATES and various error handling formats. For more information, see *Handling Errors* on page 1-53.
- Log files that preserve a record of all data that is entered onto the screen. For details, see *Logging Transactions* on page 1-58.

MODIFY also has additional screen control options such as clearing the screen, setting the height and width parameters, and changing the default size of the TYPE message area in order to enlarge the CRTFORM. See *Additional Screen Control Options* on page 1-58 for more information.

Conditional and Non-Conditional Fields

When you execute a MODIFY request, FOCUS keeps track of which transaction fields are active or inactive during execution. In order to add, update, and delete segment instances, the fields must be active.

You can define data entry and turnaround fields as either conditional or non-conditional. A conditional field is conditionally active. That is, it becomes active only if there is incoming data present for the field. Otherwise, it remains inactive. A non-conditional field is always active whether there is incoming data present or not.

When you are performing update operations, there are several important points to keep in mind when you choose whether to specify a field as conditional or non-conditional.

Reference

Usage Notes for Determining Whether to Specify a Field as Conditional or Non-Conditional

- If data is entered or changed, the data source value is always updated and the field always becomes active. This is true whether the field is conditional or non-conditional.
- If data is not entered or changed, what happens to the data source value is dependent on whether the field is conditional or non-conditional as well as program logic. The following table outlines this.

Type of Field	Syntax	Active/Inactive	Database Value	Transaction Value
Conditional Entry	<	Active (changed)	Displayed value replaces data source value.	Value.
		Inactive	Remains.	Blank.
Conditional Turnaround	<T.	Active (changed)	Displayed value replaces data source value.	Value.
		Inactive	Remains.	Blank.
Non-conditional Entry	< >	No user entry	Displayed value replaces data source value (blank or 0).	Value.
Non-conditional Turnaround	<T.>	No user entry	Displayed value replaces data source value (same value).	Value.

- If a field is active, the displayed value always becomes the new data source value. In turnaround fields, this is by definition the same value.
- If a field is inactive, the displayed value is always ignored.
- If you compute a data source field and then display it on the CRTFORM with a D. or a T., the field must still be active to get the computed value displayed on the screen. Otherwise, you get a blank or a 0.
- When you use a VALIDATE for a field, the field must be active. Otherwise you do not get the accurate data source value validated; instead, you get a blank or 0.

Note: You can make a field active or inactive by using the ACTIVATE or DEACTIVATE statement respectively.

Example

Using Conditional and Non-Conditional Display and Turnaround Fields

The following example illustrates the display and turnaround field features as well as the use of a non-conditional turnaround field (both carets). The first CRTFORM asks for a key field value, in this case EMP_ID. If a matching record is obtained, then some data source values are displayed and others are shown for turnaround update.

Note how the non-conditional turnaround field functions in the following example. Whether the displayed value is changed or not, the value in the data source is active. The VALIDATE uses the display value, whether it was changed or not.

```

MODIFY FILE EMPLOYEE
1. CRTFORM
   "ENTER EMPLOYEE ID#: <EMP_ID>"
   "PRESS ENTER BEFORE CONTINUING"
   "-----"
MATCH EMP_ID
  ON NOMATCH TYPE
    "EMPLOYEE ID NOT IN DATABASE. PLEASE REENTER."
  ON NOMATCH REJECT
2. ON MATCH CRTFORM LINE 4
   " "
   "EMPLOYEE ID #:          <D.EMP_ID>"
   "LAST NAME:             <D.LAST_NAME>"
   "HIRE DATE:             <D.HIRE_DATE>"
   "SALARY:                 <T.CURR_SAL>"
   "DEPARTMENT:           <T.DEPARTMENT>"
3. ON MATCH VALIDATE
   SALTEST = IF CURR_SAL GT 0 THEN 1 ELSE 0;
   ON INVALID TYPE
     "SALARY MUST BE GREATER THAN 0"
     "CORRECT SALARY AND PRESS ENTER TWICE"
   ON MATCH UPDATE CURR_SAL DEPARTMENT
DATA VIA FIDEL
END

```

The example processes as follows:

1. When the procedure executes, the top part of the CRTFORM displays as follows:

```
ENTER EMPLOYEE ID #:  
PRESS ENTER BEFORE CONTINUING  
-----
```

If the employee ID entered does not match an ID in the data source, the transaction is rejected and a TYPE statement appears at the bottom of the screen. Assume the operator enters the following employee ID:

```
ENTER EMPLOYEE ID #: 818692173  
PRESS ENTER BEFORE CONTINUING  
-----
```

2. If the ID entered matches an ID in the data source, FOCUS successfully retrieves a record. The ON MATCH CRTFORM causes a second CRTFORM to be displayed on line 4. This CRTFORM contains both display and turnaround fields. The data source values of the fields appear on the second CRTFORM, and the cursor is positioned at the start of the CURR_SAL field which is the first unprotected field. Note that both CURR_SAL and DEPARTMENT are automatically highlighted for turnaround:

```
ENTER EMPLOYEE ID #: 818692173  
PRESS ENTER BEFORE CONTINUING  
-----  
EMPLOYEE ID #:      818692173  
LAST NAME:          CROSS  
HIRE DATE:           811102  
SALARY:              27062.00  
DEPARTMENT:         MIS
```

Assume the operator updates DEPARTMENT, does not change CURR_SAL, and transmits the CRTFORM:

```

ENTER EMPLOYEE ID #: 818692173
PRESS ENTER BEFORE CONTINUING
-----
EMPLOYEE ID #:      818692173
LAST NAME:         CROSS
HIRE DATE:         811102
SALARY:            27062.00
DEPARTMENT:       ois

```

- When the operator presses Enter, the transaction is processed. If the value of CURR_SAL is greater than 0, the VALIDATE will evaluate as 1 (true) and processing continues. Although a value was not entered for CURR_SAL, the field is active because it is specified as a non-conditional field. Thus, the VALIDATE reads the current value in the T. field (27062.00), and validates the field. The transaction is then processed.

If you specify the turnaround field as conditional (only the left caret), the field is inactive if no data is entered. Assume the same transaction as above. The operator updates the DEPARTMENT and does not enter new data for the CURR_SAL field. The VALIDATE does not read the T. value because the field is inactive and instead reads a 0. The field is invalidated and the following error message occurs:

```

ENTER EMPLOYEE ID #: 818692173
PRESS ENTER BEFORE CONTINUING
-----
EMPLOYEE ID #:      818692173
LAST NAME:         CROSS
HIRE DATE:         811102
SALARY:            27062.00
DEPARTMENT:       ois

(FOC421)TRANS 1 REJECTED INVALID SALTEST
INVALID SALARY
SALARY MUST BE GREATER THAN 0

```

Using FIXFORM and FIDEL in a Single MODIFY

A MODIFY procedure can mix data sources from CRTFORMs and FIXFORMs.

- You can have only one FIXFORM statement and you must specify the name of the transaction data source. For example:

```
FIXFORM ON filename
```

- The FIXFORM statement must precede the CRTFORM statement.
- START and STOP do not apply.
- You must use the DATA VIA FIDEL statement for CRTFORM data input.

This feature is useful in situations where a known set of records is wanted and the keys for these records reside on an external fixed format data source. (The data source may have been produced by a prior TABLE and SAVE or HOLD command.) The procedure first reads a key, fetches the matching record, and displays it on the CRTFORM specified.

This is also convenient when the FIXFORM is included in a START case.

Example

Using FIXFORM With FIDEL

To run this example on your machine, you must first create a sequential data source with data. To do so, execute the following TABLE request:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID PAY_DATE
IF PAY_DATE GE 820730
ON TABLE SAVE AS PAYTRANS
END
```

This creates the transaction data source PAYTRANS. Then execute the following MODIFY request:

```
MODIFY FILE EMPLOYEE
1. FIXFORM ON PAYTRANS EMP_ID/9 PAY_DATE/6
2. MATCH EMP_ID
   ON NOMATCH REJECT
   ON MATCH CONTINUE
MATCH PAY_DATE
3.  ON MATCH/NOMATCH CRTFORM
    "EMPLOYEE ID #:      <D.EMP_ID>"
    "PAY DATE:          <D.PAY_DATE>"
    "MONTHLY GROSS:     <T.GROSS>"
    ON NOMATCH INCLUDE
    ON MATCH UPDATE GROSS
DATA VIA FIDEL
END
```

The example processes as follows:

1. First the data is read in from the sequential data source PAYTRANS.
2. The EMP_ID from PAYTRANS is matched against EMP_IDs in the EMPLOYEE data source. If it matches, PAY_DATE is matched.
3. The CRTFORM is displayed with display values for EMP_ID and PAY_DATE. If there is a match on PAY_DATE, GROSS is displayed as a turnaround field and the operator can update it. If there is no match on PAY_DATE, both PAY_DATE and GROSS are included:

```
EMPLOYEE ID #:      071382660
PAY_DATE:          820831
MONTHLY GROSS:     916.67
```

The procedure ends when there are no more transactions on the external data source to read. It can also be terminated by the operator pressing the PF1 or PF3 key.

Default CRTFORM Processing Within the Same MODIFY Request

All CRTFORM commands within the same MODIFY request are concatenated into a single screen image and displayed simultaneously. These CRTFORMs follow one another on the screen. The first CRTFORM is displayed on line 1, the second begins on the line after the end of the first. They are displayed immediately upon the execution of the MODIFY and are not affected by the conditions set up by MATCH logic (for example, when the MODIFY request accesses multiple segments of the data source). You can override this default by specifying the LINE option in the CRTFORM command. For additional information on the LINE option, see *Using Multiple CRTFORMs: LINE* on page 1-34.

When turnaround or display fields within a CRTFORM are dependent on MATCH logic or the value of a key field that has yet to be provided, these fields remain blank until the correct information is entered or the MATCH has been performed. Once the value of the key field is provided, the turnaround and display fields will contain the correct information.

Both of these characteristics are displayed in the following MODIFY request which contains two CRTFORMs: the first modifying the root segment, EMPINFO of the EMPLOYEE data source, and the second modifying a child segment, BANKINFO.

```
MODIFY FILE EMPLOYEE
CRTFORM
" "
"-----"
"EMPLOYEE ID #: <EMP_ID LAST NAME: <LAST NAME"
" "
"DEPARTMENT: <DEPARTMENT <28 SALARY: <CURR_SAL"
" "
"BANK: <BANK_NAME"
"FILL IN THE ABOVE FORM AND PRESS ENTER"
"-----"

MATCH EMP_ID
    ON NOMATCH REJECT
    ON MATCH UPDATE LAST_NAME DEPARTMENT CURR_SAL
    ON MATCH CONTINUE TO BANK_NAME
    ON NOMATCH INCLUDE
    ON MATCH/NOMATCH CRTFORM
    "</1"
    "FUND TRANSFER INFORMATION UPDATE"
    " "
"-----"
"BANK: <D.BN ACCT #: <T.BA"
" "
"BANK CODE: <T.BC <30 START DATE: <T.EDATE"
"-----"
    ON MATCH UPDATE BA BC EDATE
DATA VIA FIDEL
END
```

When the request is executed, both CRTFORMs display:

```

-----
EMPLOYEE ID #:                LAST NAME:

DEPARTMENT:                   SALARY:

BANK:

FILL IN THE ABOVE FORM AND PRESS ENTER
-----

FUND TRANSFER INFORMATION UPDATE

-----
BANK:                          ACCT #:

BANK CODE:                     START DATE:
-----

```

The turnaround and display fields in the second CRTFORM will not contain any values until values have been entered for the upper key. When you fill in the top part of the CRTFORM and press Enter, the screen will be refreshed and the display and turnaround fields in the second part of the CRTFORM will contain the appropriate values.

```

-----
EMPLOYEE ID #:                818692173    LAST NAME:    CROSS
DEPARTMENT:                   MIS                SALARY:       27062.00
BANK:                          BANK ASSOCIATION
FILL IN THE ABOVE FORM AND PRESS ENTER
-----

FUND TRANSFER INFORMATION UPDATE

-----
BANK:                          BANK ASSOCIATION    ACCT #: 163800144
BANK CODE:                     175963                START DATE:   830501
-----

```

At this point you can modify any of the turnaround fields for the BANKINFO segment. When you press Enter again, the transaction will be processed by FOCUS and a new composite CRTFORM is displayed.

Generating Automatic CRTFORMs

You can use several simple but powerful statements with the FOCUS MODIFY facility to allow immediate generation of data management requests. You do not need to learn the complete FOCUS MODIFY language. Without mentioning field names, you can write general-purpose requests and customize them for more detailed situations.

Syntax

How to Automatically Specify Conditional Fields

The statements can be used with multi-segment data sources as well as simple data sources. These statements automatically specify conditional fields. They include:

- | | |
|---|---|
| <code>CRTFORM * [SEG <i>n</i>]</code> | Design screen for all real data fields in segment <i>n</i> , where <i>n</i> is either the segment name or number. |
| <code>CRTFORM * KEYS [SEG <i>n</i>]</code> | Design screen for all key fields in segment <i>n</i> . |
| <code>CRTFORM * NONKEYS [SEG <i>n</i>]</code> | Design screen for all non-key fields in segment <i>n</i> . |
| <code>CRTFORM T.* [SEG <i>n</i>]</code> | Design screen using T.fields in segment <i>n</i> . |
| <code>CRTFORM D.* [SEG <i>n</i>]</code> | Design screen using D.fields in segment <i>n</i> . |

Note: The use of CRTFORM * on a COMBINE data source name is illogical and may produce unpredictable results.

How to Obtain Segment Names and Numbers

You can optionally specify the segment name or number for each of the CRTFORMs. To obtain the segment names and numbers, enter the following command where *file* is the name of the data source:

```
CHECK FILE file PICTURE
```

The names and numbers appear on the top of each segment in the diagram. You may also list segment names and numbers by entering the command:

```
? FDT filename
```

Example

Generating Automatic CRTFORMs to Add and Maintain Data

If you are modifying all the segments in the data source (except for unique segments), you can write the request without using Case Logic. The following example adds and maintains data for the EMPLOYEE data source. The segments are as follows:

- Segment 1 contains basic employee data (for example, names, jobs, salaries).
- Segment 3 contains employee salary histories.
- Segment 7 stores employees' home addresses and information on their bank accounts.

- Segment 8 stores employee monthly pay.
- Segment 9 stores monthly deductions.

(Segment 2 is a unique segment. Segments 4, 5, and 6 are cross-referenced segments that are not part of the EMPLOYEE data source.)

The request is:

```

MODIFY FILE EMPLOYEE
CRTFORM
  "THIS PROCEDURE ADDS NEW RECORDS AND UPDATES EXISTING RECORDS </1"
  "INSTRUCTIONS"
  "1. ENTER DATA FOR EACH FIELD"
  "2. USE TAB KEY TO MOVE CURSOR"
  "3. PRESS ENTER WHEN FINISHED"
  "4. WHEN YOU FINISH ALL RECORDS, PRESS PF1 </1"
CRTFORM * KEYS
MATCH * KEYS SEG 01
  ON MATCH/NOMATCH CRTFORM T.* NONKEYS SEG 01
  ON MATCH UPDATE * SEG 01
  ON NOMATCH INCLUDE
MATCH * KEYS SEG 03
  ON MATCH/NOMATCH CRTFORM T.* NONKEYS SEG 03
  ON MATCH UPDATE * SEG 03
  ON NOMATCH INCLUDE
MATCH * KEYS SEG 07
  ON MATCH/NOMATCH CRTFORM T.* NONKEYS SEG 07
  ON MATCH UPDATE * SEG 07
  ON NOMATCH INCLUDE
MATCH * KEYS SEG 08
  ON MATCH/NOMATCH CRTFORM T.* NONKEYS SEG 08
  ON MATCH UPDATE * SEG 08
  ON NOMATCH INCLUDE
MATCH * KEYS SEG 09
  ON MATCH/NOMATCH CRTFORM T.* NONKEYS SEG 09
  ON MATCH UPDATE * SEG 09
  ON NOMATCH INCLUDE
DATA VIA FIDEL
END

```

When the procedure executes, the screen displays as follows:

```
THIS PROCEDURE ADDS NEW RECORDS AND UPDATES EXISTING RECORDS

INSTRUCTIONS
1. ENTER DATA FOR EACH FIELD
2. USE TAB KEY TO MOVE CURSOR
3. PRESS ENTER WHEN FINISHED
4. WHEN YOU FINISH ALL RECORDS, PRESS PF1

EMP_ID      :      :
DAT_INC     :      :
TYPE        :      :
PAY_DATE    :      :
DED_CODE    :      :

LAST_NAME   :      :      FIRST_NAME   :      :
HIRE_DATE   :      :      DEPARTMENT   :      :
CURR_SAL    :      :      CURR_JOBCODE  :      :
ED_HRS      :      :

PCT_INC     :      :      SALARY       :      :
JOBCODE     :      :

ADDRESS_LN1 :      :
ADDRESS_LN2 :      :
ADDRESS_LN3 :      :

ACCTNUMBER  :      :
GROSS       :      :
```

Notice that the fields are divided into five groups. The first group consists of all the key fields in the data source. Each subsequent group consists of all non-key fields in a particular segment. Fill in each group from top to bottom and press Enter before filling in the next group. When you do this the request uses the values to match on the segments specified later in the request.

The first CRTFORM statement generates the first group of fields, which are all the key fields in the data source:

```
CRTFORM * KEYS
```

The MATCH statements in the request modify each of the segments in the data source. Each statement contains a CRTFORM phrase that prompts for all non-key fields in the segment:

```
CRTFORM T.* NONKEYS SEG xx
```

Note that the CRTFORM phrase displays the fields as turnaround fields. After you fill in the fields in the group and press Enter, FOCUS uses the field values to update the segment.

You can add the following enhancements to the request:

- The LINE option on each CRTFORM statement.
- Explanatory text after each CRTFORM statement.
- A separate CRTFORM containing explanatory text at the beginning of the request.

Example

Using the MATCH Statement to Add and Maintain Data

If you want to modify some but not all segments in the data source, use Case Logic to write the request. Place each MATCH statement in a separate case. The following example adds and maintains data for the EMPLOYEE data source. The segments are as follows:

- Segment 1 contains basic employee data (for example, names, jobs, salaries).
- Segment 3 contains employee salary histories.
- Segment 7 stores employees' home addresses and information on their bank accounts.
- Segment 8 stores employee monthly pay.
- Segment 9 stores monthly deductions.

(Segment 2 is a unique segment. Segments 4, 5, and 6 are cross-referenced segments that are not part of the EMPLOYEE data source.)

This request modifies data in Segments 1, 3, and 7:

```

MODIFY FILE EMPLOYEE
CRTFORM
  "THIS PROCEDURE MAINTAINS EMPLOYEE"
  "JOB DATA, SALARY HISTORIES, AND ADDRESSES"
  " "
CRTFORM * KEYS
  "FILL IN EMP_ID, DAT_INC, AND TYPE FIELDS"
  "THEN PRESS ENTER"
GOTO EMPLOYEE

CASE EMPLOYEE
MATCH * KEYS SEG 01
  ON NOMATCH REJECT
  ON MATCH CRTFORM T.* NONKEYS SEG 01 LINE 10
  ON MATCH UPDATE * SEG 01
  ON MATCH GOTO MONTHPAY
ENDCASE

```

```
CASE MONTHPAY
MATCH * KEYS SEG 03
    ON MATCH/NOMATCH CRTFORM T.* NONKEYS SEG 03 LINE 10
    ON MATCH UPDATE * SEG 03
    ON MATCH GOTO DEDUCT
    ON NOMATCH INCLUDE
    ON NOMATCH GOTO DEDUCT
ENDCASE

CASE DEDUCT
MATCH * KEYS SEG 07
    ON MATCH/NOMATCH CRTFORM T.* NONKEYS SEG 07 LINE 10
    ON MATCH UPDATE * SEG 07
    ON NOMATCH INCLUDE
ENDCASE
DATA VIA FIDEL
END
```

Using Multiple CRTFORMs: LINE

You can choose what screen line the CRTFORM will begin on by using the LINE option. By default, the first CRTFORM begins on line 1. The next CRTFORM in the procedure begins on the line following the end of the previous CRTFORM. For example, if one screen uses 12 lines, the next CRTFORM automatically begins on the 13th line.

The LINE option enables you to control both the placement of a CRTFORM on the screen and the timing with which it displays on the screen. Using LINE gives you the following options:

- You can have one logical form replace another after each transaction by having subsequent CRTFORMs begin on the same line.
- You can build mixed screens by saving lines from a previous CRTFORM on the screen while executing a subsequent CRTFORM. In other words, the first CRTFORM is displayed, the operator transmits the data, and the next CRTFORM is displayed while the previous one remains on the screen.

Syntax

How to Use the LINE Option

```
CRTFORM [LINE nn]
```

where:

nn

Is the starting line number for the CRTFORM.

Example

Using the LINE Option

In the following example there are two logical forms: EMPLOYEE UPDATE and FUND TRANSFER INFORMATION UPDATE. It illustrates the placement of CRTFORMs when the default is in effect (that is, the LINE option is not used).

```

MODIFY FILE EMPLOYEE
1. CRTFORM
  "EMPLOYEE UPDATE"
  " "
  "-----"
  "EMPLOYEE ID #:   <EMP_ID   LAST_NAME: <LAST_NAME"
  " "
  "DEPARTMENT:     <DEPARTMENT <28 SALARY: <CURR_SAL"
  " "
  "BANK:   <BANK_NAME"
  " "
  "FILL IN THE ABOVE FORM AND PRESS ENTER"
  "-----"
MATCH EMP_ID
ON NOMATCH REJECT
ON MATCH UPDATE LAST_NAME DEPARTMENT CURR_SAL
ON MATCH CONTINUE TO BANK_NAME
  ON NOMATCH INCLUDE
  ON MATCH/NOMATCH CRTFORM
2. "</1"
  "FUND TRANSFER INFORMATION UPDATE"
  " "
  "-----"
  "BANK: <D.BN   ACCT #: <T.BA"
  " "
  "BANK CODE: <T.BC <30 START DATE: <T.EDATE"
  "-----"
  ON MATCH UPDATE BA BC EDATE
DATA VIA FIDEL
END

```

This produces the following screen when the request is executed:

```
EMPLOYEE UPDATE
-----
EMPLOYEE ID #:          LAST_NAME:
DEPARTMENT:            SALARY:
BANK:
FILL IN THE ABOVE FORM AND PRESS ENTER
-----
FUND TRANSFER INFORMATION UPDATE
-----
BANK:                  ACCT #:
BANK CODE:             START DATE:
-----
```

Note that when the default is in effect, each logical form is displayed one after the other on the screen, the instant the MODIFY procedure is executed. That is, all the distinct CRTFORMs are concatenated into one visual form.

Example

Using the LINE Option to Replace a Screen

To completely replace one screen with the next, both CRTFORMs must start on the same line.

```

MODIFY FILE EMPLOYEE
1. CRTFORM
  "EMPLOYEE UPDATE"
  " "
  "-----"
  "EMPLOYEE ID #:  <EMP_ID LAST_NAME: <LAST_NAME"
  " "
  "DEPARTMENT:      <DEPARTMENT <30 SALARY: <CURR_SAL"
  " "
  "BANK:  <BANK_NAME"
  " "
  "FILL IN THE ABOVE FORM AND PRESS ENTER"
  "-----"
MATCH EMP_ID
ON NOMATCH REJECT
ON MATCH UPDATE LAST_NAME DEPARTMENT CURR_SAL
ON MATCH CONTINUE TO BANK_NAME
ON NOMATCH INCLUDE
2. ON MATCH/NOMATCH CRTFORM LINE 1
  "</1"
  "FUND TRANSFER INFORMATION UPDATE"
  " "
  "-----"
  "BANK: <D.BN ACCT #: <T.BA"
  " "
  "BANK CODE: <T.BC <30 START DATE: <T.EDATE"
  "-----"
  ON MATCH UPDATE BA BC EDATE
DATA VIA FIDEL
END

```

1. When the MODIFY procedure is executed, the following screen is displayed:

```
EMPLOYEE UPDATE
-----
EMPLOYEE ID #:   LAST_NAME:
DEPARTMENT:     SALARY:
BANK:
FILL IN THE ABOVE FORM AND PRESS ENTER
-----
```

2. After the operator enters and transmits the data, the next CRTFORM replaces the previous one on the screen:

```
FUND TRANSFER INFORMATION UPDATE
-----
BANK:           ACCT #:
BANK CODE:     START DATE:
-----
```

Generally, it is a good practice to put LINE 1 on all CRTFORMs that start a new case unless a specific screen pattern is wanted.

Example**Using the LINE Option to Create a Mixed Screen**

A combination of two or more individual CRTFORMs can occupy specific lines on one screen. To obtain a mixed screen, place the desired starting line number with the CRTFORM statement. The following example demonstrates how you can create mixed screens using the LINE option.

```

MODIFY FILE EMPLOYEE
1. CRTFORM
   "EMPLOYEE UPDATE"
   " "
   "-----"
   "EMPLOYEE ID #:   <EMP_ID LAST_NAME: <LAST_NAME"
   " "
   "DEPARTMENT:     <DEPARTMENT <30 SALARY: <CURR_SAL"
   " "
   "BANK:   <BANK_NAME"
   " "
   "FILL IN THE ABOVE FORM AND PRESS ENTER"
   "-----"

MATCH EMP_ID
ON NOMATCH REJECT
ON MATCH UPDATE LAST_NAME DEPARTMENT CURR_SAL
ON MATCH CONTINUE TO BANK_NAME
ON NOMATCH INCLUDE
2. ON MATCH/NOMATCH CRTFORM LINE 12
   "</1"
   "FUND TRANSFER INFORMATION UPDATE"
   " "
   "-----"
   "BANK: <D.BN ACCT #: <T.BA"
   " "
   "BANK CODE: <T.BC <30 START DATE: <T.EDATE"
   "-----"
ON MATCH UPDATE BA BC EDATE
DATA VIA FIDEL
END

```

Processing occurs as follows:

1. Like the preceding examples, this produces the first screen. Assume the operator enters and transmits the following data:

```
EMPLOYEE UPDATE
-----
EMPLOYEE ID #: 117593129      LAST_NAME:  JONES
DEPARTMENT:  MIS              SALARY:  18480
BANK:  STATE
FILL IN THE ABOVE FORM AND PRESS ENTER
-----
```

2. The first CRTFORM remains on the screen while the next CRTFORM is displayed on line 12:

```
EMPLOYEE UPDATE
-----
EMPLOYEE ID #: 117593129      LAST_NAME:  JONES
DEPARTMENT:  MIS              CURRENT SALARY:  18480
BANK:  STATE
-----
FUND TRANSFER INFORMATION AND UPDATE
-----
BANK:  STATE                  ACCT #:
BANK CODE:                    START DATE:
-----
```

You can save certain lines from the preceding CRTFORM while you discard others. In the previous example, if you begin the second CRTFORM on line 6, the EMP_ID and the LAST_NAME remain and the next line is the beginning of the FUND TRANSFER INFORMATION AND UPDATE.

1. Assume the operator enters and transmits data on the first CRTFORM. Part of the first logical form disappears and the second form is displayed. Thus, a new visual form is created:

```
EMPLOYEE UPDATE
-----
EMPLOYEE ID #: 117593129      LAST_NAME: JONES
FUND TRANSFER INFORMATION AND UPDATE
-----
BANK: STATE                      ACCT #: 40950036
BANK CODE: 510271                START DATE: 821101
-----
```

You can create mixed screens using the LINE option, in a variety of ways, depending on the need of the application.

CRTFORMs and Case Logic

Case Logic enables you to perform separate complete MODIFY processes in one procedure. Each of these is a case, and the procedure contains directions about which case to execute under various circumstances.

When you use the Case Logic option of the MODIFY command, you can create a pattern of many CRTFORMs.

When there are multiple CRTFORMs in a single MODIFY request, use the LINE option to specify where each CRTFORM will be displayed. With Case Logic, generally, we recommend that you use LINE 1 to replace one screen with another.

Example

Using Case Logic With CRTFORM

The following example illustrates the use of Case Logic with the CRTFORM.

```
MODIFY FILE EMPLOYEE
COMPUTE
    PFKEY/A4= ;
CRTFORM
    "TO INPUT A NEW RECORD, PRESS PF4"
    "TO UPDATE AN EXISTING RECORD, PRESS PF5"

IF PFKEY EQ 'PF04' GOTO ADD ELSE
IF PFKEY EQ 'PF05' GOTO UPP ELSE GOTO TOP;

CASE ADD
CRTFORM LINE 1
    "EMPLOYEE ID #:      <EMP_ID"
    "LAST NAME: <LAST_NAME FIRST NAME: <FIRST_NAME"
    "HIRE DATE: <HIRE_DATE"
    "DEPARTMENT:          <DEPARTMENT"
MATCH EMP_ID
    ON MATCH REJECT
    ON NOMATCH INCLUDE
ENDCASE

CASE UPP
CRTFORM LINE 1
    "EMPLOYEE ID #:      <EMP_ID"
    "DEPARTMENT:          <DEPARTMENT"
    "JOB CODE: <CURR_JOBCODE"
    "SALARY:      <CURR_SAL"
MATCH EMP_ID
    ON NOMATCH REJECT
    ON MATCH UPDATE DEPARTMENT CURR_JOBCODE CURR_SAL
ENDCASE
DATA VIA FIDEL
END
```

The first CRTFORM displays as:

```
TO INPUT A NEW RECORD, PRESS PF4
TO UPDATE AN EXISTING RECORD, PRESS PF5
```

If the operator presses PF4, the following is displayed:

```
EMPLOYEE ID #:
LAST NAME:           FIRST NAME:
HIRE DATE:
DEPARTMENT:
```

If the operator presses PF5, the following is displayed:

```
EMPLOYEE ID #:
DEPARTMENT:
JOB CODE:
SALARY:
```

Note: At the end of a MODIFY procedure, control defaults to the TOP Case.

Specifying Groups of Fields

Groups of fields (that is, more than one occurrence of the same field) can be specified on the CRTFORM in two ways:

- Specifying a field more than once on a CRTFORM.
- Using REPEAT syntax.

You can use Case Logic to process groups of fields.

Examples

Specifying Groups of Fields for Input

A group of fields may repeat on the form. For example:

```
"EMPLOYEE ID      DEPARTMENT      SALARY"
"<EMP_ID         <DPT              <CURR_SAL"
"<EMP_ID         <DPT              <CURR_SAL"
"<EMP_ID         <DPT              <CURR_SAL"
```

This reads the same data as the FIXFORM statement:

```
FIXFORM 3(EMP_ID/C9 DPT/C10 CURR_SAL/C14)
```

Repeating Groups of Fields

The following example shows the use of repeating groups for a single employee ID.

```
MODIFY FILE EMPLOYEE
CRTFORM
  "ENTER EMPLOYEE ID #: <EMP_ID"
  " "
  "ENTER PAY DATE AND GROSS PAY FOR ABOVE EMPLOYEE"
  " "
  "PAY DATE: <PAY_DATE      GROSS: <GROSS"
  "PAY DATE: <PAY_DATE      GROSS: <GROSS"
  "PAY DATE: <PAY_DATE      GROSS: <GROSS"
MATCH EMP_ID
  ON NOMATCH REJECT
  ON MATCH CONTINUE
MATCH PAY_DATE
  ON MATCH REJECT
  ON NOMATCH INCLUDE
DATA VIA FIDEL
END
```

Note: A group of repeated data fields cannot be specified on a MATCH or NOMATCH CRTFORM. They must be presented on a primary CRTFORM (that is, one not generated as a result of a MATCH or NOMATCH command).

This procedure processes as follows:

```
ENTER EMPLOYEE ID #: 818692173

ENTER PAY DATE AND GROSS AMOUNT FOR ABOVE EMPLOYEE

PAY DATE: 850405      GROSS: 3000.00
PAY DATE: 850412      GROSS: 4000.00
PAY DATE: 850418      GROSS: 2500.00
```

When the operator presses Enter, the transaction processes. Processing continues until a line with no data is found or all lines are completed, whichever comes first.

Using REPEAT to Display Multiple Records

You can display multiple segment instances on the screen by directing FIDEL to read and display the contents of a HOLD buffer.

Syntax

How to Identify an Instance in the Hold Buffer Using a Subscript Value

field(n)

where:

field

Is the name of a previously held field.

n

Is the integer subscript that identifies the number of the instance in the HOLD buffer containing the field to be displayed. *n* must be in integer format or the report group will be ignored.

The variable SCREENINDEX allows you to display and modify selected groups of records from the HOLD buffer.

Example

Using the REPEAT Statement to Display Multiple Records

The following example uses the REPEAT statement to retrieve up to a set number (in this case, six) of multiple instances, saves them in the HOLD buffer, and then displays the instances on the CRTFORM.

```

MODIFY FILE EMPLOYEE
1. CRTFORM
    "PAY HISTORY UPDATE"
    " "
    "ENTER EMPLOYEE ID#: <EMP_ID"
MATCH EMP_ID
    ON NOMATCH REJECT
    ON MATCH GOTO COLLECT

CASE COLLECT
2. { REPEAT 6 TIMES
    { NEXT PAY_DATE
    { ON NEXT HOLD PAY_DATE GROSS
3. { ON NONEXT GOTO DISPLAY
    { ENDREPEAT
    GOTO DISPLAY
    ENDCASE

CASE DISPLAY
    IF HOLDCOUNT EQ 0 GOTO TOP;
4. COMPUTE
    BUFFNUMBER/I5 = HOLDCOUNT;
5. CRTFORM LINE 5
    "FILL IN GROSS AMOUNT FOR EACH PAY DATE"
    " "
    "PAY DATE          GROSS AMOUNT"
    "-----          -"
    "<D.PAY_DATE(1)    <T.GROSS(1)>"
    "<D.PAY_DATE(2)    <T.GROSS(2)>"
    "<D.PAY_DATE(3)    <T.GROSS(3)>"
    "<D.PAY_DATE(4)    <T.GROSS(4)>"
    "<D.PAY_DATE(5)    <T.GROSS(5)>"
    "<D.PAY_DATE(6)    <T.GROSS(6)>"
    GOTO UPDATE
    ENDCASE

CASE UPDATE
6. REPEAT BUFFNUMBER
    MATCH PAY_DATE
    ON NOMATCH REJECT
    ON MATCH UPDATE GROSS

ENDREPEAT
GOTO COLLECT
ENDCASE

DATA VIA FIDEL
END
    
```

The procedure processes as follows:

1. When the procedure is executed, the first CRTFORM is displayed:

```
PAY HISTORY UPDATE
ENTER EMPLOYEE ID #:
```

2. Assume the operator enters the following ID and transmits the data:

```
ENTER EMPLOYEE ID #: 071382660
```

If there is a match, the instruction is to REPEAT the logic six times. That is, up until six times, find a PAY_DATE and hold the PAY_DATE and the GROSS in the HOLD buffer.

3. When there are no more PAY_DATE fields or six of them have been held, the procedure branches to CASE DISPLAY.
4. The procedure stores the number of records that are in the HOLD buffer in the variable BUFFNUMBER.

5. The procedure displays the following CRTFORM:

```
PAY HISTORY UPDATE
ENTER EMPLOYEE ID #: 071382660
FILL IN GROSS AMOUNT FOR EACH PAY DATE

PAY DATE      GROSS AMOUNT
820831        916.67
820730        916.67
820630        916.67
820528        916.67
820430        916.67
820331        916.67
```

The operator makes changes to the fields in the GROSS AMOUNT column and presses Enter. All changes for all records are transmitted simultaneously as shown:

```
PAY HISTORY UPDATE
ENTER EMPLOYEE ID #: 071382660
FILL IN GROSS AMOUNT FOR EACH PAY DATE

PAY DATE      GROSS AMOUNT
820831        816.67
820730        816.67
820630        816.67
820528        916.67
820430        916.67
820331        916.67
```

6. The REPEAT statement instructs FOCUS to perform the MODIFY logic on all segment instances.

Note: If a CRTFORM screen with subscripted variables is rejected with a FORMAT ERROR, you may not alter any records on the screen prior to the record rejected, as FOCUS has already held them.

Using Groups of Fields With Case Logic

Case Logic can be used to process a group of fields. For additional information on Case Logic, see *CRTFORMs and Case Logic* on page 1-41.

Reference

Usage Notes When Using Case Logic to Process a Group of Fields

- Each time the procedure enters the case, the next group of fields is processed. FOCUS keeps track internally of which groups have been processed.
- If the CRTFORM with the group of fields is not in the TOP case, you must create your own branching logic to process all the groups before going back to the TOP. This normally needs some kind of counting mechanism. Once the procedure goes back to the TOP case, all unprocessed data on the CRTFORM or in a lower case is lost.

Example

Using Case Logic With Groups of Fields

The following example shows the CURSORINDEX being used in conjunction with a VALIDATE. For more information on CURSORINDEX, see *Specifying Cursor Position* on page 1-17.

```

MODIFY FILE EMPLOYEE
1. CRTFORM
    "EMPLOYEE SALARY AND DEPARTMENT UPDATE"
    " "
    "PRESS ENTER"
GOTO COLLECT

CASE COLLECT
2. REPEAT 6 TIMES
    NEXT EMP_ID
        ON NEXT HOLD EMP_ID CURR_SAL DEPARTMENT
        ON NONEXT GOTO DISPLAY
ENDREPEAT
GOTO DISPLAY
ENDCASE

```

```

CASE DISPLAY
3. IF HOLDCOUNT EQ 0 GOTO EXIT;
4. COMPUTE
    BUFFNUMBER/I5 = HOLDCOUNT;
5. CRTFORM LINE 7
    "EMPLOYEE           SALARY           DEPARTMENT"
    "-----"         "-----"         "-----"
    "<D.EMP_ID(1)       <:AAA.T.CSAL(1)>   <:BBB.T.DPT(1)>"
    "<D.EMP_ID(2)       <:AAA.T.CSAL(2)>   <:BBB.T.DPT(2)>"
    "<D.EMP_ID(3)       <:AAA.T.CSAL(3)>   <:BBB.T.DPT(3)>"
    "<D.EMP_ID(4)       <:AAA.T.CSAL(4)>   <:BBB.T.DPT(4)>"
    "<D.EMP_ID(5)       <:AAA.T.CSAL(5)>   <:BBB.T.DPT(5)>"
    "<D.EMP_ID(6)       <:AAA.T.CSAL(6)>   <:BBB.T.DPT(6)>"
6. REPEAT 6 TIMES
    COMPUTE
        CURSOR/A66 = ':AAA';
        CURSORINDEX/I5=REPEATCOUNT;
    VALIDATE
        SALTTEST = IF CSAL GT 50000 THEN 0 ELSE 1;
        ON INVALID TYPE "SALARY MUST BE LESS THAN $50,000"
        ON INVALID GOTO DISPLAY
    ENDREPEAT
    GOTO UPDATE
    ENDCASE

CASE UPDATE
7. REPEAT BUFFNUMBER
    MATCH EMP_ID
        ON NOMATCH REJECT
        ON MATCH UPDATE CURR_SAL DEPARTMENT
    ENDREPEAT
    GOTO COLLECT
    ENDCASE
DATA VIA FIDEL
END
```

The example processes as follows:

1. The first CRTFORM requests the operator to press Enter without typing anything.
2. The REPEAT statement retrieves six employee IDs, salaries, and department assignments and places them in a buffer.
3. If there are no records in the buffer, the procedure terminates.
4. The COMPUTE statement stores the number of records in the buffer in the variable BUFFNUMBER.
5. The second CRTFORM retrieves the IDs, salaries, and department assignments from the buffer and displays them together on the screen. Note the field labels:

The label :AAA on the CURR_SAL (CSAL) field.

The label :BBB on the DEPARTMENT (DPT) field.

Assume that the operator changes the values to the following:

EMPLOYEE SALARY AND DEPARTMENT UPDATE		
PRESS ENTER		
EMPLOYEE	SALARY	DEPARTMENT
-----	-----	-----
071382660	35000.00	PRODUCTION
112847612	23200.00	MIS
117593129	75480.00	MIS
119265415	19500.00	PRODUCTION
119329144	39700.00	PRODUCTION
123764317	36862.00	PRODUCTION

- The second REPEAT statement operates on each of the six records displayed by the second CRTFORM, in order of display, performing the following tasks:

Sets the CURSOR variable to the label :AAA.

Sets the CURSORINDEX variable to the number of the record it is processing (1 through 6).

Validates the CURR_SAL field value. If the CURR_SAL value is \$50,000 or more, the procedure branches back to the beginning of Case DISPLAY. The procedure displays the second CRTFORM again, with the CURSOR and CURSORINDEX variables positioning the cursor on the invalid salary.

In the example, the procedure positions the cursor on the third CURR_SAL value:

```
EMPLOYEE SALARY AND DEPARTMENT UPDATE

PRESS ENTER

EMPLOYEE           SALARY           DEPARTMENT
-----           -
071382660          35000.00        PRODUCTION
112847612          23200.00        MIS
117593129          _75480.00       MIS
119265415          19500.00        PRODUCTION
119329144          39700.00        PRODUCTION
123764317          36862.00        PRODUCTION

(FOC421)TRANS 2 REJECTED INVALID SALTEST
SALARY MUST BE LESS THAN $50,000
```

- If all values are valid, the third REPEAT statement updates the employee's salary and department for each record in the buffer. The procedure then branches to Case COLLECT to update six more records in the data source.

Handling Errors

It is important to know how various errors are handled by FIDEL so that proper instructions can be given to terminal operators. Following are several kinds of errors that can cause a transaction or screen of data to be rejected:

- A format error caused by entering non-numeric data for a numeric field.
- A validation error caused by entering an incoming value that failed a VALIDATE test coded in the MODIFY.
- A NOMATCH condition caused by entering data for a key field that did not match any record in the data source.
- A DUPLICATE condition caused by key field values that matched records in the data source.
- An ACCEPT error caused by entering a value for a data source field which failed the ACCEPT test.

Handling Format Errors

If the operator enters a non-numeric character into a field defined as numeric, an error message is displayed and the screen is not processed (processing stops). The error message indicates the line number and field name in error and the cursor is automatically positioned on that field. Additionally, if the operator enters a value that fails an ACCEPT test for a field an error message is displayed and the screen is not processed. Any message specified for that field with the HELPMESSAGE attribute will also be displayed.

The operator can retype the data and press the Enter key to retransmit the screen. Alternatively, the operator may press the PF2 key to cancel the transaction. The error prevents anything on the screen from being processed. When the operator corrects the error and transmits the screen, processing resumes.

There are two exceptions to this rule. When there are repeating groups, all complete transactions up to the error will be processed. Also, in REPEAT/HOLD loops, the data prior to the format error may not be altered.

VALIDATE and CRTFORM Display Logic

When the operator enters a value that is invalid, the transaction is rejected and an error message is displayed on the screen. By default, control returns to the first CRTFORM in the TOP case. However, you can use an ON INVALID GOTO statement to transfer control to any other case in the request.

If the NOCLEAR or blank option in the CRTFORM statement is in effect, the screen will not be cleared. The operator can change the data in the offending transaction and retransmit the screen.

When you use validations, you can divide the tests into different cases and repeat a case if it fails the test. The advantage of this is that the operator can change the invalid data and retransmit the screen before other sections are processed. An ON INVALID TYPE phrase can be used to send an informative message to the operator on the screen.

Example

Using the ON VALID TYPE Phrase

The following example shows how you can use the ON VALID TYPE phrase to send an informative message to the operator on the screen.

```
CASE TRY
CRTFORM
  "EMPLOYEE ID #: <EMP_ID NAME: <LAST_NAME"
  "CURRENT SALARY: <CURR_SAL"
VALIDATE
  GOODSAL= CURR_SAL GT 10000 AND CURR_SAL LT 1000000;
  ON INVALID TYPE
  "THE CURRENT SALARY CANNOT BE LARGER THAN 1000000 OR"
  "LESS THAN 10000"
  ON INVALID GOTO TRY
  .
  .
  .
```

All messages appear on the bottom four lines of the screen, unless you specify the TYPE option on the CRTFORM statement. For more information on the TYPE option, see *Additional Screen Control Options* on page 1-58.

Handling Errors With Repeating Groups

If old style repeating groups (those without subscripts) are present and there is an error, processing continues to the next transaction on the screen. This means that if the operator changes the offending transaction and retransmits the screen, the other transactions on the screen become duplicates. It is important when using repeating groups to reject duplicates and turn the duplicate message off (LOG DUPL MSG OFF).

Alternatively, avoid using VALIDATE with repeating groups. Use COMPUTE instead and branch to a case that displays the erroneous data in a lower portion of the screen.

Example

Using COMPUTE to Display Erroneous Data

In the following example, a test field is computed in Case TEST, using DECODE. This test field checks that the department value is a valid one. If the operator inputs a department value that is invalid, control branches to a case that displays the erroneous data (CASE BADDPT).

```

MODIFY FILE EMPLOYEE
1. CRTFORM
  "FILL IN THE FOLLOWING CHART WITH THE SALARIES"
  "AND DEPARTMENT ASSIGNMENTS OF FOUR NEW EMPLOYEES"
  " "
  "          EMPLOYEE ID      DEPARTMENT      SALARY"
  "          -----      -"
  "PERSON 1  <EMP_ID          <DEPARTMENT    <CURR_SAL"
  "PERSON 2  <EMP_ID          <DEPARTMENT    <CURR_SAL"
  "PERSON 3  <EMP_ID          <DEPARTMENT    <CURR_SAL"
  "PERSON 4  <EMP_ID          <DEPARTMENT    <CURR_SAL"
  GOTTO TEST

2. CASE TEST
  IF EMP_ID IS ' ' GOTO TOP;
  COMPUTE
    TEST/I1 = DECODE DEPARTMENT (MIS 1 PRODUCTION 1 ELSE 0);
  IF TEST IS 0 GOTO BADDEPT ELSE GOTO ADD;
  ENDCASE

4. CASE ADD
  MATCH EMP_ID
  ON NOMATCH INCLUDE
  ON MATCH REJECT
  ENDCASE

3. CASE BADDEPT
  COMPUTE
    XEMP/A9 = EMP_ID;
    XDEPT/A10 = DEPARTMENT;
  CRTFORM LINE 12
    "INVALID ENTRY: DEPARTMENT MUST BE MIS OR PRODUCTION"
    "CORRECT THE ENTRY BELOW"
    " "
    "EMPLOYEE ID: <D.XEMP      DEPARTMENT: <T.XDEPT"
  COMPUTE
    DEPARTMENT=XDEPT;
  GOTTO TEST
  ENDCASE

DATA VIA FIDEL
END

```

The request processes as follows:

1. This is the first and TOP case and contains a CRTFORM that displays four instances of repeating groups. Assume the operator fills in values and transmits the screen. Control transfers to Case TEST.
2. Case TEST contains a computed field that uses DECODE to make sure that the values that have been input for DEPARTMENT are either MIS or PRODUCTION. When a DEPARTMENT value does not match this list, TEST is returned a code of 0, in which case control transfers to Case BADDEPT.
3. Case BADDEPT first computes two fields, XEMP and XDEPT, to have the values of EMP_ID and DEPARTMENT at the time the error occurred. Next, BADDEPT displays a CRTFORM containing a message to the operator and the two computed fields. The XDEPT field, which contains the invalid DEPARTMENT value, is a turnaround field so that the operator can see the invalid value and change it. Next, the COMPUTE is reversed and the new values are returned to their respective fields. Control transfers back to Case TEST where the DEPARTMENT values will continue to be tested until they are all valid. At that point, control transfers to Case ADD.
4. Case ADD contains the MATCH logic necessary to include new employees into the EMPLOYEE data source. The transaction including all the repeating groups is processed at one time.

Rejecting NOMATCH or Duplicate Data

When the request directs that transactions be rejected, an error message is displayed on the screen. It is a good idea, when the major keys do not repeat, to use a split CRTFORM and give the keys in the first CRTFORM. Once the keys are accepted, the rest of the data may be entered.

Example

Using NOMATCH to Return Control to Operator

In the following example, if the EMP_ID does not match, control returns immediately to the operator with a request to correct the value. If a match does occur, the operator must then fill in the balance of the form and transmit it.

```

MODIFY FILE EMPLOYEE
CRTFORM
  "ENTER EMPLOYEE ID#:      <EMP_ID"
  "THEN PRESS ENTER "
MATCH EMP_ID
  ON NOMATCH TYPE
    "ID NOT IN DATABASE  PLEASE REENTER"
  ON NOMATCH REJECT
  ON MATCH CRTFORM LINE 4
    "LAST NAME:      <T.LAST_NAME"
    "DEPARTMENT:    <T.DEPARTMENT"
    "SALARY:        <T.CURR_SAL"
  ON MATCH UPDATE LAST_NAME DEPARTMENT CURR_SAL
DATA VIA FIDEL
END

```

If repeating groups are present and no other cases are involved, all of the groups are processed before control returns to the screen. Thus, splitting screens in this way is particularly useful when the second CRTFORM contains repeating groups.

Logging Transactions

You can log the data entered on the screen to any log file. Only the data is logged, not the CRTFORM, so a compact log file is created.

Examples

Logging Data to a Log File

```
LOG TRANS ON ALLDATA
```

This will log transactions to a file allocated to the ddname ALLDATA.

The record length of the file must allow space for each field on each CRTFORM in the procedure, plus one character at the start of each CRTFORM. The record length should not be longer than this.

This may be an inconvenient format since it is very long if several CRTFORMs exist. Instead you can construct a custom log file of your own design using TYPE statements.

Constructing a Custom Log File

The following example logs data collected from its preceding CRTFORM to a file allocated to ddname MYCRT, including a COMPUTE transaction number, TNUM.

```
CRTFORM
  "EMPLOYEE ID #: <EMP_ID  NAME <LAST_NAME"
  "HIRE DATE: <HIRE_DATE"
COMPUTE
  TNUM/I4=TNUM+1;
  TYPE ON MYCRT
"<TNUM><EMP_ID><LAST_NAME><HIRE_DATE"
```

This option is recommended, rather than the standard LOG option, whenever a procedure contains more than two CRTFORMs, or when text or computed fields need to be captured on the log file.

Additional Screen Control Options

MODIFY CRTFORMs support several additional screen control options:

- Clearing the screen with CLEAR/NOCLEAR.
- Specifying the screen size with WIDTH/HEIGHT.
- Changing the size of the message area at the bottom of the screen using TYPE. This increases the length of the screen that can be used for the actual form.

Clearing the Screen: CLEAR/NOCLEAR

Data is transmitted from the CRTFORM to the data source when the operator presses the Enter key. After each successful screen is processed, the data areas are automatically cleared. You can override this default by using the NOCLEAR option. Then, after each data transmission, the screen remains unchanged.

This is a useful feature when there is a substantial amount of data that carries over from one screen to another.

Syntax

How to Clear the Screen Using CLEAR/NOCLEAR

CRTFORM action

where:

action

Is one of the following:

blank is the default. Causes the screen to clear after the data is transmitted. If a transaction is invalid and an error message appears at the bottom of the screen, the screen will not be cleared.

NOCLEAR causes the data values on the screen to remain as is after data is transmitted.

CLEAR causes the data values on the screen to clear after every data transmission, even if there is an error. Thus, if CLEAR is specifically used and there is an error, data must be reentered.

Note: The options chosen may be different from one CRTFORM to the next.

Index

C

conditional fields, 1-21
 FIDEL, 1-21

CRTFORM

- automatic, 1-30
- Case Logic, 1-41
- conditional fields, 1-21
- cursor position, 1-17
- display fields, 1-9
- display logic, 1-53
- FIXFORM command, 1-26
- groups of fields, 1-43, 1-54
- handling errors, 1-53
- labeled fields, 1-15
- LINE command, 1-34
- log file, 1-58
- lowercase option, 1-8
- mixed screens, 1-26
- non-conditional fields, 1-21
- PF keys, 1-13
- spot markers, 1-6
- turnaround fields, 1-9
- VALIDATE command, 1-53

cursor position
 CRTFORM, 1-17

CURSORAT command
 CRTFORM, 1-18

CURSORINDEX command
 CRTFORM, 1-17

D

data entry fields
 FIDEL, 1-9

defining fields in FIDEL, 1-15

display fields, 1-9

E

embedded data
 FIDEL, 1-6

F

FIDEL, 1-1

- automatic CRTFORMs, 1-30
- COMPUTE command, 1-21
- conditional fields, 1-21
- cursor position, 1-17
- field positioning, 1-6
- FIXFORM command, 1-26
- format errors, 1-9
- labeling fields, 1-15
- MODIFY, 1-21
- non-conditional fields, 1-21
- PF keys, 1-13
- repeating groups, 1-45
- screen elements, 1-4
- screen operating conventions, 1-2
- spot markers, 1-6
- syntax, 1-4

field attributes
 FIDEL, 1-5

FIXFORM command
 FIDEL, 1-26

FOCUS Interactive Data Entry Language, 1-1

- automatic CRTFORMs, 1-30
- COMPUTE command, 1-21
- conditional fields, 1-21
- cursor position, 1-17
- field positioning, 1-6
- FIXFORM command, 1-26
- format errors, 1-9
- labeling fields, 1-15
- MODIFY, 1-21
- non-conditional fields, 1-21
- PF keys, 1-13
- repeating groups, 1-45

- screen elements, 1-4
- screen operating conventions, 1-2
- spot markers, 1-6
- syntax, 1-4

L

- labels
 - FIDEL, 1-5, 1-15
- LINE command
 - CRTFORM, 1-34
- log file
 - CRTFORM, 1-58

M

- MODIFY, 1-1, 1-2
 - FIDEL, 1-21

N

- non-conditional fields, 1-21
 - FIDEL, 1-21

P

- PF keys (FIDEL), 1-13

- branching, 1-14
 - default settings, 1-13
 - query command, 1-13
 - resetting, 1-13

R

- repeating fields
 - FIDEL, 1-43
- repeating groups
 - FIDEL, 1-45, 1-54

S

- spot markers
 - FIDEL, 1-6

T

- turnaround fields
 - FIDEL, 1-9

V

- VALIDATE command
 - FIDEL, 1-21

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. Send comments to:

Corporate Publications
Attn: Manager of Documentation Services
Information Builders
Two Penn Plaza
New York, NY 10121-2898

or FAX this page to (212) 967-0460, or call Elizabeth Soudant at (212) 736-4433, **x3819**.

Name: _____

Company: _____

Address: _____

Telephone: _____ Date: _____

Comments:

Reader Comments