

Burroughs

**Reference
Manual**

**B 20 Systems
Forms**

(Relative to Release Level 4.0)

*Priced Item
Printed in U.S.A.
July 1984*

1168549

**Reference
Manual**

**B 20 Systems
Forms**

(Relative to Release Level 4.0)
Copyright © 1984, Burroughs Corporation, Detroit, Michigan 48232

**Priced Item
Printed in U.S.A.
July 1984**

1168549

Burroughs cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded using the Remarks form at the back of the manual, or may be addressed directly to Corporate Documentation-West, Burroughs Corporation, 1300 John Reed Court, City of Industry, California 91745, U.S.A.

LIST OF EFFECTIVE PAGES

Page	Issue
Title	Original
ii	Original
iii	Original
iv	Blank
v thru x	Original
1-1 thru 1-6	Original
2-1 thru 2-23	Original
2-24	Blank
3-1 thru 3-6	Original
4-1 thru 4-22	Original
A-1 thru A-6	Original
B-1 thru B-5	Original
B-6	Blank
C-1 thru C-11	Original
C-12	Blank
D-1 thru D-5	Original
D-6	Blank
1 thru 3	Original
4	Blank

TABLE OF CONTENTS

Section	Title	Page
	INTRODUCTION	ix
1	OVERVIEW	1-1
	Installing the B 20 Forms Utility	1-1
	Creating and Editing Forms (Forms Editor).	1-2
	Files, Forms, and Fields	1-2
	Field Definitions	1-3
	Name	1-3
	Repeating Fields	1-3
	Default	1-4
	Highlights	1-4
	Auto-exit During Program Run	1-4
	Running a Report on a Form (Forms Reporter).	1-4
	Running a Program Using Your Form (Forms Run Time)	1-5
	Opening a Form	1-6
	At Run Time	1-6
	At Link Time	1-6
	Variable Naming Convention	1-6
2	FORMS EDITOR	2-1
	Accessing the Forms Editor to Create A Form	2-1
	Accessing the Forms Editor to Modify A Form	2-1
	Status and Work Areas	2-3
	Moving the Cursor	2-4
	Cursor Key Plus Shift	2-4
	Cursor Key Plus Code	2-4
	Cursor Keys in Combination	2-4
	Selections	2-5
	Captions	2-6
	Edit Modes	2-6
	Delete and Back Space Keys	2-7
	Literal Insert	2-7
	Commands	2-7
	Reselect (f1)	2-8
	Undo (f2)	2-8
	Draw (f4)	2-9
	Drawing Lines	2-9
	Drawing Boxes	2-9

TABLE OF CONTENTS (Cont)

Section	Title	Page
	Erase (f5)	2-10
	Read Form (f6)	2-10
	Write Form (f7)	2-12
	Define Field (f8)	2-13
	Defining Single Fields	2-13
	Defining Repeating Fields	2-16
	Modifying Field Definitions	2-18
	Modifying Field Position and Width	2-18
	Test Drive (f9)	2-19
	Delete Selection (f10)	2-19
	Copy	2-20
	Move	2-21
	Zoom (Code Z)	2-22
	View Edit Codes	2-22
	Next Tag	2-22
	Exiting the Forms Editor	2-23
3	FORMS REPORTER	3-1
4	FORMS RUN TIME	4-1
	Variable Names	4-1
	Prefixes	4-2
	Roots	4-3
	Suffixes	4-3
	Variable Name Examples	4-4
	Forms Run Time Services	4-4
	Run Time Sequence	4-5
	DefaultField	4-5
	DefaultForm	4-6
	DisplayForm	4-6
	GetFieldInfo	4-7
	LockKbd	4-9

TABLE OF CONTENTS (Cont)

Section	Title	Page
	OpenForm	4-9
	ReadField	4-10
	SetFieldAttrs	4-10
	UndisplayForm	4-11
	UserFillField	4-11
	WriteField	4-14
	Error Response	4-14
	Type Codes	4-15
	Configuring the Type System	4-16
	Forms Run Time Structure	4-16
	FmRgtd.asm Source Text	4-17
	Defining Types	4-18
	Starter Kit of Types	4-18
	Adding Types	4-19
	Range Checking and Data Validation	4-22
A	ERROR MESSAGES	A-1
	Forms Editor Error Messages	A-1
	Forms Run Time Error Messages	A-4
B	SAMPLE BASIC PROGRAM	B-1
C	TRAINING EXERCISE: CREATE A FORM	C-1
	Access the Forms Editor to Create A Form	C-1
	Start With Captions	C-2
	Add some Lines and Boxes	C-3
	Add Tabular Columns	C-6
	Define Fields for User Entry	C-7
	Additional Training Activities	C-11
D	GLOSSARY OF TERMS	D-1
	INDEX	1

LIST OF ILLUSTRATIONS

Figure	Title	Page
2-1	Forms Editor Command Form	2-2
2-2	Read Form Command Form	2-10
2-3	Define Field Command Form	2-13
2-4	Fields Tags (Sample)	2-15
2-5	Save Form	2-23
3-1	FReport Command Form	3-1
3-2	Sample Form Report	3-3
C-1	Selected Caption Box	C-3
C-2	Captions In A Box	C-3
C-3	Selected Horizontal Line	C-4
C-4	Subdivided Box	C-5
C-5	Tabular Form	C-6
C-6	Selection For A Repeating Field	C-8
C-7	Defined Fields	C-9
C-8	Finished Form	C-10

LIST OF TABLES

Table	Title	Page
2-1	Forms Editor Commands	2-8
2-2	Character Attributes	2-15
4-1	Variable Prefixes	4-2
4-2	Variable Root Terms	4-3
4-3	Variable Suffixes	4-3
4-4	Forms Run-Time Services	4-5
4-5	cbFieldInfoRet and cbFieldInfoMax Format	4-8
4-6	pInitState and pExitStateRet Information	4-13
4-7	Conversion Procedure Module	4-18

INTRODUCTION

This manual contains descriptive and procedural information on the B 20 Forms Utility: Forms Editor, Forms Reporter, and Forms Run Time.

You should be familiar with standard B 20 Executive commands. You also need some programming knowledge to run or link a form to a program (but programming knowledge is not required to create or design a form).

This manual is organized in sections, as follows:

- Section 1, Overview, provides general information on using the B 20 Forms utility, including software installation, the **FORMS EDITOR**, the Forms Reporter, and Forms Run Time.
- Section 2, Forms Editor, contains procedures for creating, editing, and testing a form, using the **FORMS EDITOR**.
- Section 3, Forms Reporter, contains procedures for displaying a report on a form.
- Section 4, Forms Run Time, describes Forms Run Time services, error response, and type codes, and provides information for structuring a custom type system.

This manual also contains four appendixes, as follows:

- Appendix A lists error and status messages and discusses possible causes and solutions.
- Appendix B contains a BASIC program which uses the Forms utility.

Appendix C contains a training exercise with step-by-step instructions for constructing a form and displaying a report on the form.

Appendix D contains a glossary of terms.

Additional related information is available in the following manuals (referenced in sections of this manual):

- B 20 Systems Custom Installation and Reference Manual*
- B 20 Systems Linker/Librarian Reference Manual*
- B 20 Systems Operating System (BTOS) Reference Manual*
- B 20 Systems Standard Operations Guide*

SECTION 1

OVERVIEW

The B 20 Forms utility consists of:

- an editor (**FORMS EDITOR** command) to help you design forms
- a forms reporter (**FREPORT** command) to display information about a form
- run time modules (Forms Run Time) which you can link to a program to display forms and accept data supplied by a user

Throughout this manual, user refers to the operator of a program.

INSTALLING THE B 20 FORMS UTILITY

If you have a hard disk B 20 system, you install the B 20 Forms utility software by using the **SOFTWARE INSTALLATION** command and the Forms System Disk. The **FORMS EDITOR**, Forms Reporter, and Forms Run Time are then available at the Executive. (For more information on the **SOFTWARE INSTALLATION** command, refer to the *B 20 Systems Custom Installation and Reference Manual*.)

If you have a dual floppy B 20 system, each time you use the **FORMS EDITOR**, Forms Reporter, or Forms Run Time services you must insert the Forms System Disk in drive f0 (after you sign on). Insert the floppy disk for the form you are creating or editing in drive f1.

CREATING AND EDITING FORMS (FORMS EDITOR)

You design or edit forms (draw lines, type captions, and define user entry fields) using the **FORMS EDITOR**. Since Forms Run Time permits a program to refer to fields symbolically, you can often edit a form without changing the program.

While you are editing or designing the form, the **FORMS EDITOR** displays it as it will appear at program run. The **FORMS EDITOR** also has a command (**TEST DRIVE**), that allows you to fill in fields as a user. When you **TEST DRIVE** or **WRITE** your form, the **FORMS EDITOR** compacts the form into a set of tables. When you finish editing the form, you can save the form (or the latest version of the form) in a file to be called at run time.

Files, Forms, and Fields

Forms are filed in special form files (a B 20 file with a suffix, .form, automatically added by the **FORMS EDITOR**). Forms consist of lined and captioned displays with fields for accepting user data entries or displaying computed data. A form can contain many nonoverlapping fields, limited by screen size and byte availability of your system. You define the fields when you create or change a form.

Each form has a name, separate from its file name. If the form is the only form in the file, the form name and file name are usually the same.

When a form is stored in a library file with other forms, the form name is used to distinguish the form from the other forms. These forms are stored in object module format; groups of forms can therefore be bundled together using the **LIBRARIAN** utility (described in the *B 20 Systems Linker/Librarian Reference Manual*).

Each field also has a distinguishing name; however, you can also designate repeating fields (many fields with the same name). Repeating fields are distinguished by their index number.

Field Definitions

Lines and captions are protected fields; user input at run time cannot change them. You define unprotected fields by using the **DEFINE FIELD** command of the **FORMS EDITOR**. Defined fields have field names and, optionally, default values that will or will not display for the user, indexes (if repeating fields), highlights when selected or unselected, and automatic exit when the user enters a last character.

Name

The field name is a text string up to 40 characters in length. When a program calls Forms Run Time, it specifies the field name (and index if the field is repeating). Field names must be unique, unless all fields that share the name are repeating. The distinction between uppercase and lowercase is ignored for matching field names.

Repeating Fields

Repeating fields are groups of fields that have the same name. Tabular forms with columns containing rows of data usually have repeating fields. Each column contains many fields of the same name.

The repeating fields are distinguished by their indexes (location). If the repeating field is vertical, the **FORMS EDITOR** automatically assigns index numbers consecutively (starting with 1). However, you can assign index numbers when you define the repeating field (**DEFINE FIELD** command of the **FORMS EDITOR**). No two fields (locations) can have the same field name and index number.

You can modify the number of fields for repeating fields without changing the program that calls it; however, the program must use the Forms Run Time Service `GetFieldInfo` (described in section 4) to determine the number of fields.

Default

Each field has a default value that the field is set to when the form is first displayed. The default is empty (null) unless you enter a value when you define (or redefine) the field.

Unless you set **Show Default?** to no when you define a field, the default displays when the form is used. The default value does not have to display to be read by the program.

Highlights

You can apply visual highlights (character attributes) to fields when you define them. These character attributes make the defined field blink, display in reverse video, display half-bright, and/or display underlined.

Each field can be assigned a selected and an unselected character attribute. When a user positions the cursor in a field, the field is selected; when a user positions the cursor outside the field, the field is unselected.

Auto-Exit During Program Run

Forms Run Time returns control to the program when the user attempts to exit the field. If you set **Auto-exit?** to yes when you define a field, Forms Run Time also returns control to the program when a text character is typed in the last character cell of the field.

RUNNING A REPORT ON A FORM (FORMS REPORTER)

The Forms Reporter (**FREPORT** command) displays information on a form (form name, size, height and width, and number of fields) and lists each field's defined characteristics (name, size, single or repeating, index, and options). This report can be written to a disk file or printer.

RUNNING A PROGRAM USING YOUR FORM (FORMS RUN TIME)

Forms Run Time is a library of object module procedures. You can write a program in any B 20 programming language and use Forms Run Time services as follows:

1. The program specifies the form name and, optionally, a screen location for the display.
2. Forms Run Time uses the tables stored by the **FORMS EDITOR** to display the form.
3. The user completes fields in a sequence controlled by the program.
4. Forms Run Time prompts the user to enter data into each field and returns the data to the calling program.
5. Forms Run Time controls user exiting of each field as follows:

If the user presses a key that moves the cursor within the field, (such as **BACK SPACE**, **DELETE**, **Left Arrow** and **Right Arrow**) Forms Run Time does not return control to the program.

If the user presses keys that move the cursor from the field (such as **TAB**, **NEXT**, **Up Arrow** and **Down Arrow**), Forms Run Time exits the field and returns control to the program.

If the user types text (alphanumerics and punctuation) Forms Run Time does not return control to the program unless **Auto-exit?** is turned on (set to yes) and the text entry is in the last character cell of a field.

6. Forms Run Time encodes field input values as it returns them to the program.

Opening a Form

The forms for a program that uses Forms Run Time are typically stored in one or more files and must be opened by a program before they can be used. Forms can be opened at run time or link time, but a program is easier to maintain if the form and run files are separate.

An open form consists of a memory work area that contains all the information needed to display the form and to access its fields. The work area required is included in the Forms Reporter (**FREPORT** command) report and displays at the top of **FORMS EDITOR** screen (unless the form is full screen) so you can monitor form size as you create a form. A full-screen form requires a work area of approximately 2K.

At Run Time

The program can open the form at run time if you use **OpenForm** (described in section 4).

At Link Time

Since forms are stored as object modules, you can use the **LINKER** utility for static inclusion of the forms in the run file. The program can then refer to the form by its form name (declared as a public symbol in the form file). The form is conceptually opened at link time and you do not need **OpenForm**. (Refer to the *B*20 Systems Linker/Librarian Reference Manual*).

Variable Naming Convention

The variable names used in procedure definitions, fields of request blocks, and other data structures allow you to infer some of the variable's characteristics from its name.

Some common variable name prefixes, roots, and suffixes are listed in section 4.

SECTION 2

FORMS EDITOR

You can use the **FORMS EDITOR** to design or modify a form. The form displays as it will appear during program run. Appendix C contains a training exercise for creating a form.

ACCESSING THE FORMS EDITOR TO CREATE A FORM

To access the **FORMS EDITOR**, proceed as follows:

1. Type *forms editor* at a command prompt (B 20 Executive).
2. Press **GO**. The words **Forms Editor** appear at the top of the screen. The area below the double line is work space for creating a form.

NOTE

You can name your form (and save it) by using the **WRITE FORM** command. Your new form is not saved until you use this command.

ACCESSING THE FORMS EDITOR TO MODIFY A FORM

To access the **FORMS EDITOR** and modify a form, proceed as follows:

1. Type *forms editor* at a command prompt (B 20 Executive) and then press **RETURN** to display the **FORMS EDITOR** command form (as shown in figure 2-1).

Command forms editor
Forms Editor
File
[Form]

Figure 2-1. Forms Editor Command Form

2. Type the file name in the **File** field.

It is not necessary to add the .form suffix to the file name, as that is the default. (The .form suffix is added as part of the **WRITE FORM** command, discussed later in this section.)

The volume and directory default is the current path. If you want to access a form from a different volume or directory, you must type in the complete file name using brackets--[volume]<directory>filename.

3. If the file has more than one form, type the form name in the **[Form]** field. The default is the file name.
4. Press **GO**; the form appears.

NOTE

You can save the changed version of your form by using the **WRITE FORM** command. The changed version is not saved until you use this command.

STATUS AND WORK AREAS

A double line divides the **FORMS EDITOR** screen into two parts:

- the area above the double line is for status.

The words **Forms Editor** appear at the top left corner and messages display in the center. After the form has been stored (**WRITE FORM** command), the number of bytes the form uses appears at the top right corner and the form name appears in the center.

If you create or access a form that uses the entire screen area, the work area expands and the status information no longer displays.

- the area below the double line is the work area.

When you access the **FORMS EDITOR** to create a form, the work area is empty except for the cursor which is in the center of the screen.

When you access the **FORMS EDITOR** to modify a form, the form appears in the work area with the cursor at the last line edited.

If you create or access a form that uses the entire screen, the work area covers the status area.

Moving the Cursor

You can move the cursor anywhere on the display. You move the cursor by using the cursor keys (**Up Arrow**, **Down Arrow**, **Left Arrow**, and **Right Arrow**). The cursor moves one space in the corresponding direction (up, down, right, left) for each time the cursor key is pressed. If you hold the cursor key down, the cursor moves repeatedly.

Cursor Key Plus Shift

When you press a cursor key and the **SHIFT** key simultaneously, the cursor jumps several spaces in the corresponding direction (up, down, right, or left).

Cursor Key Plus Code

When you press a cursor key and the **CODE** key simultaneously, the cursor moves to the corresponding edge of the screen (top, bottom, right, or left).

Cursor Keys in Combination

You can center the cursor horizontally by simultaneously pressing the **Left Arrow** and **Right Arrow** keys. You can center the cursor vertically by simultaneously pressing the **Up Arrow** and **Down Arrow** keys.

You can move the cursor diagonally by simultaneously pressing the **Up Arrow** or **Down Arrow** key and the **Left Arrow** or **Right Arrow** key (by also pressing the **CODE** key, you can quickly move the cursor to a corner of the screen).

Selections

A selection is a rectangular area of the screen that you choose (select) by using the **MARK** and **BOUND** keys. After you select an area, you can use the **FORMS EDITOR** commands to draw lines or boxes, erase lines or boxes, move or copy part of the form, assign character attributes (highlights) to the selected area, and define fields.

You can select only one area at a time; however, the selection can be any size and can include fields or only a portion of a field. If you make a new selection, you automatically remove a previous selection.

To make a selection, proceed as follows:

1. Position the cursor at a corner of the rectangular area you want to select.
2. Press **MARK**. The character cell containing the cursor appears in reverse video. (You have selected this character cell.)
3. Move the cursor to another corner of the rectangular area you want to select.
4. Press **BOUND**. A reverse video rectangle appears. (You have selected this rectangle.)

You can change the width or height of your selection by pressing **BOUND** again at a different screen position. The marked corner of the selection remains fixed, but the rectangle changes to reflect the new corner.

If you press **MARK** again, you remove the previous selection and only the character cell currently containing the cursor is selected.

You can set character attributes for the selection by pressing **CODE** and a letter for the character attribute. (Refer to Define Field, later in this section, for a list of character attributes.)

Captions

Captions are protected; they cannot be overwritten by lines during forms editing or modified by the user at run time.

You can place captions anywhere, but lines and captions cannot occupy the same character cell. Two or more distinct captions can occupy the same line of a form, for example:

left caption right caption

You should separate these same line captions by cursor movement (empty character cells)--not by spaces. Spaces are characters and you cannot draw lines through character cells occupied by spaces.

If you edit one of these same line captions, the position of other captions is not affected. For example:

left abcde caption right caption

If you edit one of these same line captions such that the caption collides with the caption to the right, the system emits an audio signal. If you try to insert more characters than can fit on that line, the system emits an audio signal.

A caption overrides a selection. If you start typing a caption after you have made a selection, the selection disappears. You can recover the selection after you finish typing the caption by pressing f1 (RESELECT).

Edit Modes

You can use either insert or overwrite to add or modify captions.

In overwrite mode, the characters you type replace the characters at the cursor position. If the **OVER TYPE** key light (LED) is on, your B 20 is in overwrite mode. If the key light is off, press the **OVER TYPE** key to enter the overwrite mode.

In insert mode, the characters you type are inserted at the current cursor position (the character at the cursor position and the cursor move to the right). If the **OVER TYPE** key light is off, your B 20 is in insert mode. If the key light is on, press the **OVER TYPE** key to enter the insert mode.

NOTE

Text does not wrap. If an insert would cause the text to pass the edge of the screen or to run into text which ends at the screen edge, the system beeps and will not accept the entry.

Delete And Back Space Keys

The **DELETE** and **BACK SPACE** keys can help you edit captions. When you press **DELETE**, you remove one character (at the cursor position). **BACK SPACE** functions differently depending on which edit mode you use, as follows:

- In overtype, **BACK SPACE** is equivalent to the **Left Arrow** key: it moves the cursor to the left but does not delete text.
- In insert, **BACK SPACE** moves the cursor to the left, deleting the text to the left of the cursor position as it moves.

Literal Insert

Press **CODE** and ' (single quote) to literally insert the next key you press. All characters in the range 00h-FFh are accessible from the keyboard using this method.

COMMANDS

Access the **FORMS EDITOR** commands by pressing function keys as described in table 2-1.

Table 2-1. Forms Editor Commands

Key	Command	Use This Command To:
f1	RESELECT	Select the last selected area
f2	UNDO	Delete the last change
f4	DRAW	Draw a line or box through the selected area
f5	ERASE	Remove a line or box from the selected area
f6	READ FORM	Display a stored form
f7	WRITE FORM	Store a form
f8	DEFINE FIELD	Define field characteristics
f9	TEST DRIVE	Fill in fields as a user
f10	DELETE SELECTION	Remove captions, lines, and fields from the selection

Reselect (f1)

In most cases, when you make a selection and then use a command on that selection (for example, **DRAW**), the selection disappears after the command. If you want to select the same area of the form again, press f1 (**RESELECT**).

Undo (f2)

You can delete your last command by pressing f2 (**UNDO**). **UNDO** restores deleted text or erased lines, deletes new lines or insertions or copied portions, and puts moved material back. The selection and cursor are also returned to the positions they had occupied. Any uninterrupted sequence of text-entry operations counts as a single command for **UNDO**.

Draw (f4)

Using **DRAW**, you can draw lines or boxes on your form. Any text in the path of a line (including spaces) is unaffected; therefore, the resulting line may be broken. To draw lines or boxes, you select a portion of the screen and then press:

- **f4** to draw a line
- **SHIFT** and **f4** to draw a thick line
- **CODE** and **f4** to draw a double line

Drawing Lines

To draw a vertical line, select a rectangle with a width equal to one character cell. To draw a horizontal line, select a rectangle with a height equal to one character cell.

Drawing Boxes

To draw a box, select a rectangle with width and height greater than one character cell. After you press **f4**, four lines are drawn through the center of the character cell at each edge of the selection.

To subdivide this box with lines, make a selection for a vertical or horizontal line that ends at the box boundaries. The kind of line (thin, thick, double) does not have to correspond to the box lines.

Occasionally the correct line character is not available for an intersection. This happens because it would require 256 line font characters to do all intersections correctly, leaving no room in the character set for regular characters. When the correct line character is unavailable, the **FORMS EDITOR** supplies an approximation with the correct shape but the incorrect kind of line (thin, thick double) for a branch. If you later perform a **DRAW** at that intersection such that a correct line character becomes available, the **FORMS EDITOR** supplies the correct character.

Erase (f5)

You can use **ERASE** to remove lines or boxes. Captions within the selected area are not removed.

To erase lines or boxes, select the line, box or portion of line or box to be erased. Press f5 (**ERASE**). All the lines within the boundary of your selected area are erased. If the erased line intersects other lines, the erasure clears the intersections.

Read Form (f6)

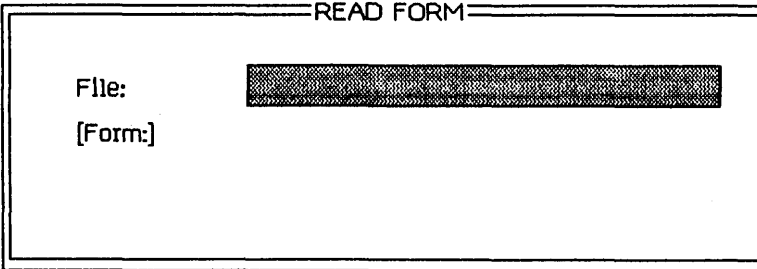
You can open a stored form as you access the **FORMS EDITOR** by pressing **RETURN** after you type the command and filling in the file and/or form name. To recall a stored form while you are in the **FORMS EDITOR**, use the **READ FORM** command (f6).

CAUTION

When the form displays, it overwrites the screen, erasing any data on the display. If it overwrites a form that you have not saved (by using **WRITE FORM**), the form is lost.

Proceed as follows:

1. Press f6 (**READ FORM**). A **READ FORM** command form displays as shown in figure 2-2. If you currently have a form on your display, that file name is in the form.



READ FORM

File:

[Form:]

Figure 2-2. Read Form Command Form

2. Type the file name in the **File:** field.

It is not necessary to add the **.form** suffix to the file name, as that is the default. (The **.form** suffix is added as part of the **WRITE FORM** command, discussed later in this section.)

The volume and directory default is the current path. If you want to access a form from a different volume or directory, you must type in the complete file name using brackets--[volume]<directory>filename.

3. If the file has more than one form, press **NEXT** to move to the **[Form:]** field. Type the form name. The default is the file name.
4. Press **GO**. If you have not saved the currently displayed form, the **FORMS EDITOR** prompts you to confirm you wish to erase it. If you want to erase the form and display the form you requested, press **GO**.

If you press **CANCEL** instead of **GO**, the **READ FORM** command is terminated.

You can use the **READ FORM** command to recall the previous version of a form you have been editing (if you have not yet stored the edited form under that form name).

- If you want to keep your new version and display the old, press **f7 (WRITE FORM)** and store your present form under a new form name. Then use **READ FORM** to display the earlier version.
- If you want to return to the earlier version (deleting your changes), press **f6 (READ FORM)** while the form is displayed. A message displays, requesting you to confirm that you want to delete the currently displayed form. Press **GO** again to proceed with the **READ FORM** command or press **CANCEL** to terminate the command.

Write Form (f7)

You can save and name a form while it is displayed by using the **WRITE FORM** command. Form files are written as object modules to enable form bundling using the **LIBRARIAN** utility and static inclusion of forms into programs using the **LINKER**. Like any object module, a form file has a module name (the form name). If you are using the **LIBRARIAN** to package forms into a library, do not rename form files prior to calling the **LIBRARIAN** or Forms Run Time will not function correctly.

NOTE

If you are using a B 21, it is important that you frequently save (**WRITE FORM**) your form. If you exceed the character attribute limitation, the **FORMS EDITOR** exits and all changes since the last **WRITE FORM** are lost.

Proceed as follows:

1. Press f7 (**WRITE FORM**). A **WRITE FORM** command form displays.
2. Type the file name in the File field.

The volume and directory default is the current path. If you want to store the form in a different volume or directory, you must type in the complete file name using brackets--[volume]<directory>filename.

3. Press **GO**; the message **Writing...** appears. The **FORMS EDITOR** adds a .form suffix to the file name and translates the form into the binary format of a form file. Status information appears in the status area unless the form is too large.

If you press **CANCEL** instead of **GO**, the **WRITE FORM** command is terminated.

Define Field (f8)

You can use the **DEFINE FIELD** command to add or redefine fields and display the properties of individual fields.

Defining Single Fields

You can define a single field for any rectangular area that is exactly one character cell high. Proceed as follows:

1. Select the area using the cursor and the **MARK** and **BOUND** keys.
2. Press **f8** (**DEFINE FIELD**). A **DEFINE FIELD** command form displays (as shown in figure 2-3).

DEFINE FIELD

Name: Index:

Default value:

Options:
Show default? Auto-exit? Repeating?

Attributes: Unselected: Selected:

Figure 2-3. Define Field Command Form

3. Use the **NEXT** key to move the cursor to the next field or use the **Up Arrow** key to move the cursor to the previous field. Respond to the fields, as follows:

- 1) **Field Name.** You must type a distinguishing name for the field (maximum 40 characters).
- 2) **Index.** Leave this field blank. The index is used to identify repeating field locations.
- 3) **Default Value.** Type a default value (alpha or numeric) if desired.

The default is an empty field (null). If the default you enter occupies more character cells than the field size, the **FORMS EDITOR** trims the default value (from the right) to fit.

- 4) **Show Default?** If you do not want the default to show during run time, use the **DELETE** key to remove the **Yes** and type no.
- 5) **Auto-exit?** If you want the auto exit, use the **DELETE** key to remove the **No** and type yes.
- 6) **Repeating?** Leave repeating set to **No** since this is a single field.

If **Repeating?** is set to yes, your selection was for more than one field. Press **CANCEL** to terminate **DEFINE FIELD** and then change your selection, or follow the Repeating Fields procedure (later in this section).

- 7) **Attributes.** The defaults are **A** (no highlighting) for unselected and **E** (reverse video) for selected. If you want to change the character attribute, use the **DELETE** key to remove the letter and type a new entry. (Table 2-2 lists the available character attributes.)

Defining Repeating Fields

Usually repeating fields are vertically stacked; however, you can align fields horizontally or distribute them randomly.

You can define vertically stacked repeating fields in one operation. For horizontal or random fields, you can define the fields vertically and then use **MOVE** (discussed later in this section) to position each field or you can define each repeating field separately.

Proceed as follows to define a repeating field:

1. Select the field using the cursor and the **MARK** and **BOUND** keys.

For vertically stacked repeating fields, select all fields.

For horizontal or random repeating fields, select the first field.

2. Press **f8 (DEFINE FIELD)**. A **DEFINE FIELD** command form displays (as shown in figure 2-3). Use the **NEXT** key to move the cursor to the next field in the form or use the **Up Arrow** key to move the cursor to the previous field.
3. Respond to the name and index fields, as follows:

- 1) **Field Name.** You must type a distinguishing name for the field (maximum 40 characters). The same field name must be used for each field.

- 2) **Index.**

For vertically stacked fields, leave the index as is (1) or change it to start index numbering from a different number.

For horizontal or random fields, enter an index number as you define each field. Start with the field in the top left of the form

and number each field sequentially, moving right and then down. The index is used to identify each field's location.

4. The other field characteristics are optional. For vertically stacked fields, each field will initially be given the same characteristics. You can later select each field and modify these characteristics.
 - 1) Default Value. Type a default value (alpha or numeric) if desired.

The default is an empty field (null). If the default you enter occupies more character cells than the field size, the **FORMS EDITOR** trims the default value (from the right) to fit.
 - 2) Show Default? If you do not want the default to show during run time, use the **DELETE** key to remove the **Yes** and type no.
 - 3) Auto-exit? If you want the auto exit, use the **DELETE** key to remove the **No** and type yes.
 - 4) Repeating? If you selected an area for vertically stacked repeating fields, the **Repeating?** option is set to **Yes**. If you selected one field of a repeating field, use the **DELETE** key to remove the **No** and type yes.
 - 5) Attributes. The defaults are **A** (no highlighting) for unselected and **E** (reverse video) for selected. If you want to change the character attribute, use the **DELETE** key to remove the letter and type a new entry. (Table 2-2 lists the available character attributes.)
5. Press **GO**. The form reappears. A square box appears for each character cell in each defined field. These square boxes are called a tag. If you define two adjacent fields (with no separating captions or lines), one field contains filled boxes. (Tags are illustrated in figure 2-4.)

If you press **CANCEL** instead of **GO**, you terminate the **DEFINE FIELD** command. The form reappears.

6. If you are creating horizontal or random fields, move each field to its location (being careful to keep the indexes sequential, left to right then top to bottom) or repeat steps 1 through 5 to define each field.

Modify Field Definitions

You can use the **DEFINE FIELD** command to modify existing fields. You can select a single square box of a defined field to redefine the field. Your selection is automatically adjusted to include the entire field. Proceed as follows:

- **Single Fields.** Select the field. Press **f8 (DEFINE FIELD)**. Edit the **DEFINE FIELD** form. Press **GO**.
- **Vertically Aligned Repeating Fields.** Select the field (select the entire field height to modify all fields, for example, to renumber the index). Press **f8 (DEFINE FIELD)**. Edit the **DEFINE FIELD** form. Press **GO**.
- **Horizontal or Random Repeating Fields.** Select each field. Press **f8 (DEFINE FIELD)**. Edit the **DEFINE FIELD** form. Press **GO**.

Modify Field Position and Width

You can move the field or change its width by editing the field's tag, as follows:

- Insert spaces in front of the tag to move the field to the right, or backspace in front of the tag to move the field to the left.
- Delete some of the tag boxes using to shorten the field (and move it to the left).
- You can also move or copy fields (**MOVE** and **COPY** are discussed later in this section).

Test Drive (f9)

The **TEST DRIVE** command simulates the field responses of a form as it appears during a program run. Proceed as follows:

1. Press **f9 (TEST DRIVE)**. The **FORMS EDITOR** translates the form into the binary format of a form file (however, no file is written). The field tags are removed and the cursor is positioned within the first field. **Forms Run Time** controls the form.
2. You can now try out your form by typing text and using **NEXT** to move the cursor through the fields. As you exit each field, **Forms Run Time** reads your entry and displays the information. You can verify the character attributes, show default, and auto-exit. If you press a key that is unknown to **Forms Run Time**, a message displays.
3. Press **f9 (TEST DRIVE)** again to return control to the **FORMS EDITOR**. Field tags reappear.

Delete Selection (f10)

You can use the **DELETE SELECTION** command to delete (erase) all captions, lines, and fields from the area you have selected. If you split the tag in two, the first part of the tag becomes the field and the second part becomes a caption whose appearance is a sequence of square boxes. Proceed as follows:

1. Select the area to be deleted (use the cursor and the **MARK** and **BOUND** keys).
2. Press **f10 (DELETE SELECTION)**. The selected area is completely replaced with empty spaces.

NOTE

You can also press **CODE** and **DELETE** to access the **DELETE SELECTION** command. Use **f5 (ERASE)** to delete only lines, not text.

Copy

You can copy lines, fields, and captions from one part of the form to another by selecting the area to be copied and using the **COPY** key. Proceed as follows:

1. Select the area to be copied (the source area).
2. Position the cursor at the top left corner of the area to receive the copy (the target area).

Source and target areas can overlap.

3. Press **COPY**. The selected source area is copied to the target area you specified. Lines extending across the edges of the copy are trimmed or joined to the surrounding lines.

To copy the source area to a target area specified by placing the cursor at the top right corner of the target area, use the **SHIFT** and **COPY** keys.

To copy the source area to a target area specified by placing the cursor at the top center of the target area, use the **CODE** and **COPY** keys.

After **COPY**, the target area remains selected, and the cursor moves to the same position relative to the new selection as it had relative to the old. Press **COPY** a second time to make three copies of the source area, spaced equally across the screen.

CAUTION

If you **COPY** fields, the field must be repeating or you must either redefine the field to be repeating or rename one of the fields. If you **TEST DRIVE** a form with duplicate single field names, the **FORMS EDITOR** assigns the blinking character attribute to all fields in conflict.

Move

You can move lines, fields, and captions from one part of the form to another by selecting the area to be moved and using the **MOVE** key. Proceed as follows:

1. Select the text, lines, or fields to be moved (the source area).
2. Position the cursor at the top left corner of the area to receive the moved text, lines, or fields (the target area).

Source and target areas can overlap.

3. Press **MOVE**. The selected source area is moved to the target area you specified. Lines extending across the edges of the areas are trimmed or joined to the surrounding lines.

To move the source area to a target area specified by placing the cursor at the top right corner of the target area, use the **SHIFT** and **MOVE** keys.

To move the source area to a target area specified by placing the cursor at the top center of the target area, use the **CODE** and **MOVE** keys.

After **MOVE**, the target area remains selected, and the cursor moves to the same position relative to the new selection as it had relative to the old. Press **MOVE** a second time to move the original source area by twice the distance in the same direction.

CAUTION

If you move fields, be careful that repeating fields are still sequential. Edit the index numbers if they are not. If you **TEST DRIVE** a form with non-sequential fields, the **FORMS EDITOR** assigns the blinking character attribute to all fields in conflict.

Zoom (Code Z)

ZOOM is only available on B 22 systems.

The **ZOOM** command changes the display from 132 to 80 columns, and back again. Press **CODE** and **Z** to switch modes. If the screen was formatted in 132-column mode, it is changed to 80-column mode, and vice versa.

If the form in the Work Frame is more than 80 columns wide, **ZOOM** displays the message **Form is too wide** and does not change the display.

View Edit Codes

CAUTION

Literal insert spaces are lost when you use **CODE** and **V** because the command does not distinguish between spaces made visible with a previous use of the command and space characters entered via the literal insert command.

This command makes all formatting visible on the screen. Press **CODE** and **V** to view formatting. Space Bar characters show as dots. Press **CODE** and **V** again to exit.

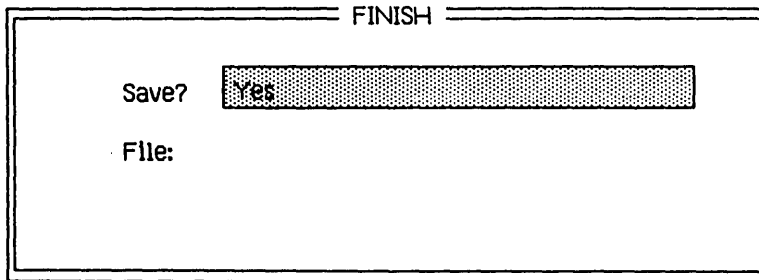
Next Tag

You can press **NEXT** to select the next field tag, an easy way to move through a form you are editing. If you press **NEXT**, your B 20 starts at the current cursor position and scans row by row down the screen, searching for a tag. If the cursor is currently in a tag, the search begins just beyond that tag. If no tag is found between the cursor and the bottom of the screen, the search resumes at the top of the screen.

When a tag is found, it is selected, and the cursor positioned under the first square box of that tag. You can edit that tag or press **NEXT** again to move on.

EXITING THE FORMS EDITOR

Press **FINISH** to end a session with the **FORMS EDITOR** and return your B 20 to the Executive. If you have not saved the form you edited or created (using **WRITE FORM**), a Save form displays similar to figure 2-5.



The image shows a rectangular dialog box with a double-line border. At the top center, the word "FINISH" is written. Below it, on the left, is the label "Save?". To the right of "Save?" is a rectangular dropdown menu with a stippled background and the word "Yes" inside. Below "Save?" is the label "File:". To the right of "File:" is an empty rectangular text input field.

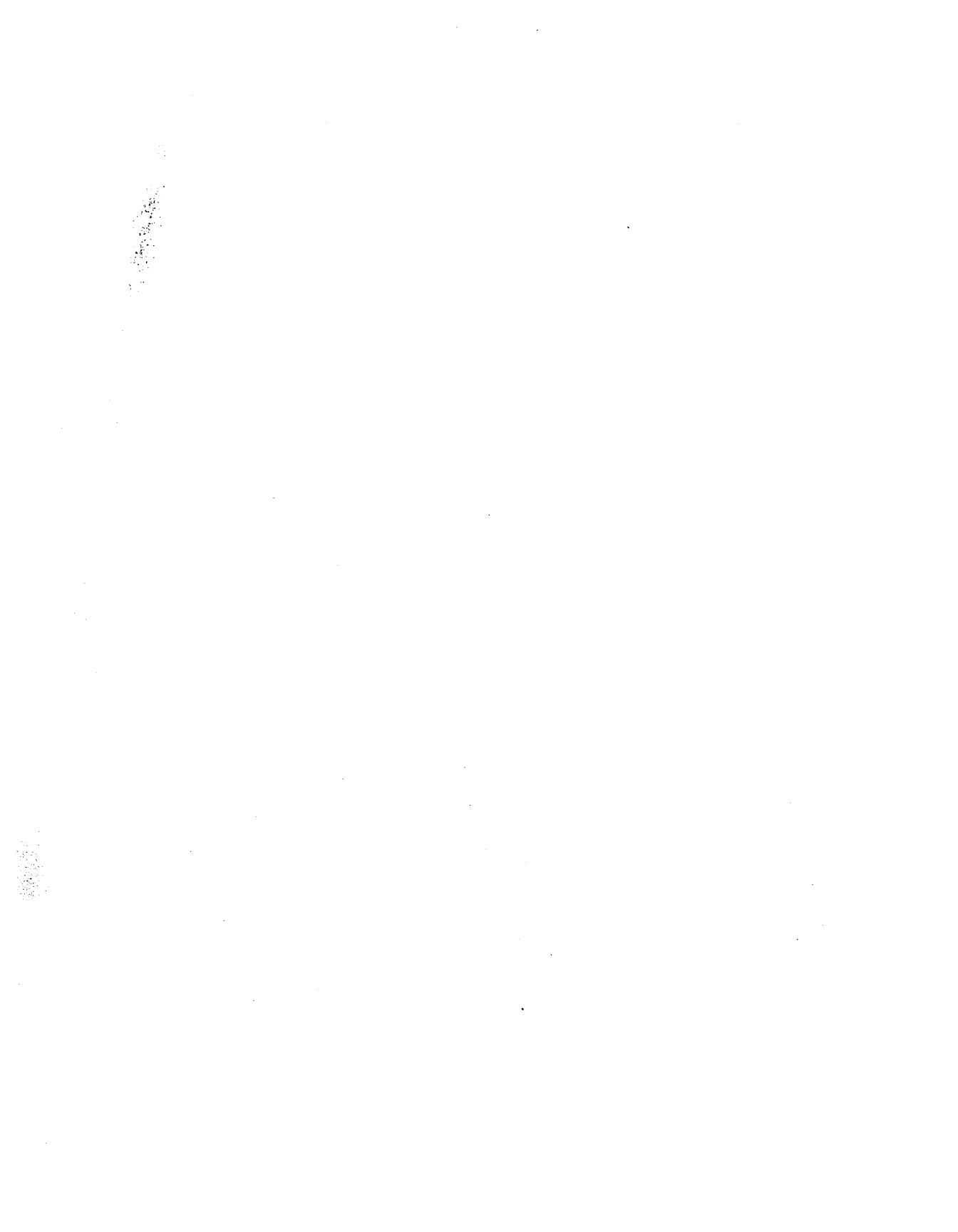
Figure 2-5. Save Form

The **File:** field default is the previous file name. Your options are:

- To save the form in the named file, press **GO**.
- To save the form in a different file, edit the **File** field and press **GO**.
- To delete the form, edit the **Save?** field to No, and press **GO**.
- To terminate **FINISH**, press **CANCEL**.

NOTE

If you have a dual floppy system, reinsert a System Disk in drive f0 to return to the Executive.



SECTION 3

FORMS REPORTER

The Forms Reporter (command **FREPORT**) displays the following form information: the form name, the size (bytes), the displayed height and width, and the number of defined fields.

In addition, by default the Forms Reporter displays the following information for each field of the form: the field name, the row and column number (the top left corner has the coordinates 0,0), the width in characters, whether or not the field is repeating, the field index, the first and last indexes for the field, the field default value, the setting for show default and auto-exit, and the field's character attributes when selected and unselected.

The report can be written to a disk file or printer.

To display a forms report, proceed as follows:

1. Type *freport* at a command prompt (Executive).
2. Press **GO**. The **FREPORT** command form appears (as shown in figure 3-1).

```
Command freport
FReport

File
[Form]
[Fields?]
[Output]
```

Figure 3-1. FReport Command Form

3. You must type the file name in the **File** field.

It is not necessary to add the .form suffix to the file name, as that is the default. (The .form suffix is added as part of the **WRITE FORM** command, discussed in section 2.)

The volume and directory default is the current path. If you want to access a form from a different volume or directory, you must type in the complete file name using brackets--[volume]<directory>filename).

4. The other fields of the **FREPORT** command form are optional, as follows:

- 1) **[Form]**. Type a form name if the specified file is a library file containing more than one form. The default is the file name.
- 2) **[Fields?]**. If you do not want information reported for the fields, type no. The default is yes.
- 3) **[Output]**. Type a device name, if desired. For example, type [lpt] to print the report. The default is to only display the report.

If your form has many fields, the report takes more than one screen to display. For this reason, you may want to print the form report or specify a file to store it in.

5. Press **GO**. A forms report displays similar to figure 3-2. You may have to press the **NEXT PAGE** or **SCROLL UP** keys to display the entire form. When the last item of the report is displayed, the Forms Reporter exits to the Executive.

Form name: Tutorial size: 904 bytes
height: 16 width: 36 number of fields: 22

Field name: Salesman
Row: 0 Column: 12 Width: 23
Repeating? No Index: (first: last:)
Default:
Show default? Yes Auto-exit? No Unselected: C Selected: E

Field name: PartNumber
Row: 4 Column: 1 Width: 10
Repeating? Yes Index: 1 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: Quantity
Row: 4 Column: 12 Width: 7
Repeating? Yes Index: 1 (first: 1 last: 5)
Default: 1
Show default? No Auto-exit? No Unselected: A Selected: E

Field name: UnitPrice
Row: 4 Column: 20 Width: 7
Repeating? Yes Index: 1 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: TotalPrice
Row: 4 Column: 28 Width: 7
Repeating? Yes Index: 1 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: PartNumber
Row: 6 Column: 1 Width: 10
Repeating? Yes Index: 2 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Figure 3-2. Sample Form Report (Sheet 1 of 4)

Field name: Quantity
Row: 6 Column: 12 Width: 7
Repeating? Yes Index: 2 (first: 1 last: 5)
Default: 1
Show default? No Auto-exit? No Unselected: A Selected: E

Field name: UnitPrice
Row: 6 Column: 2: 7
Repeating? Yes Index: 2 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: TotalPrice
Row: 6 Column: 28 Width: 7
Repeating? Yes Index: 2 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: PartNumber
Row: 8 Column: 1 Width: 10
Repeating? Yes Index: 3 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: Quantity
Row: 8 Column: 12 Width: 7
Repeating? Yes Index: 3 (first: 1 last: 5)
Default: 1
Show default? No Auto-exit? No Unselected: A Selected: E

Field name: UnitPrice
Row: 8 Column: 20 Width: 7
Repeating? Yes Index: 3 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: TotalPrice
Row: 8 Column: 28 Width: 7
Repeating? Yes Index: 3 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Figure 3-2. Sample Form Report (Sheet 2 of 4)

Field name: PartNumber
 Row: 10 Column: 1 Width: 10
 Repeating? Yes Index: 4 (first: 1 last: 5)
 Default:
 Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: Quantity
 Row: 10 Column: 12 Width: 7
 Repeating? Yes Index: 4 (first: 1 last: 5)
 Default: 1
 Show default? No Auto-exit? No Unselected: A Selected: E

Field name: UnitPrice
 Row: 10 Column: 20 Width: 7
 Repeating? Yes Index: 4 (first: 1 last: 5)
 Default:
 Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: TotalPrice
 Row: 10 Column: 28 Width: 7
 Repeating? Yes Index: 4 (first: 1 last: 5)
 Default:
 Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: PartNumber
 Row: 12 Column: 1 Width: 10
 Repeating? Yes Index: 5 (first: 1 last: 5)
 Default:
 Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: Quantity
 Row: 12 Column: 12 Width: 7
 Repeating? Yes Index: 5 (first: 1 last: 5)
 Default: 1
 Show default? No Auto-exit? No Unselected: A Selected: E

Field name: UnitPrice
 Row: 12 Column: 20 Width: 7
 Repeating? Yes Index: 5 (first: 1 last: 5)
 Default:
 Show default? Yes Auto-exit? No Unselected: A Selected: E

Figure 3-2. Sample Form Report (Sheet 3 of 4)

Field name: TotalPrice
Row: 12 Column: 28 Width: 7
Repeating? Yes Index: 5 (first: 1 last: 5)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Field name: AmountDue
Row: 14 Column: 28 Width: 7
Repeating? No Index: (first: last:)
Default:
Show default? Yes Auto-exit? No Unselected: A Selected: E

Figure 3-2. Sample Form Report (Sheet 4 of 4)

NOTE

The report shown in figure 3-2 is for the form created during the tutorial in appendix B.

SECTION 4

FORMS RUN TIME

Forms Run Time is a library of object module procedures. Programs written in any Burroughs programming languages can use Forms Run Time as follows:

- The program specifies the form name and, optionally, a screen location for the display.
- Forms Run Time uses the tables previously constructed by the **FORMS EDITOR** to display the form, prompts the user to enter data, and returns the data to the calling program.
- Fields are filled in by the user in a sequence chosen by the program.
- Field information is encoded and/or validated and returned to the program.

Appendix B contains a BASIC program that uses Forms Run Time.

VARIABLE NAMES

The variable names used in Forms Run Time procedure definitions, fields of request blocks, and other data structures allow you to infer some of the variable's characteristics from its name.

Each variable name is composed of up to three parts: prefix, root, and suffix.

NOTE

Numbers are decimal except when suffixed with "h" (for hexadecimal). Thus, 10h = 16 and 0FFh = 255.

Prefixes

The prefix identifies the data type (as shown in table 4-1). Prefixes can be compound, for example:

- cb count of bytes (the number of bytes in a string of bytes)
- pb pointer to (logical memory address of) a string of bytes
- rgb array of bytes.

Table 4-1. Variable Prefixes

b	byte (8-bit character or unsigned number)
c	count (unsigned number)
f	flag (TRUE = 0FFh or FALSE = 0)
i	index (unsigned number)
n	number (unsigned number) (same as "c")
o	offset from the segment base address (16 bits)
p	logical memory address (pointer) (32 bits consisting of the offset and the segment base address)
q	quad (32-bit unsigned integer)
rg	array of ...
s	size in bytes (unsigned number)

Roots

The root is unique to that variable or a compound of two. Some roots are listed in table 4-2.

Table 4-2. Variable Root Terms

dh	device handle
erc	status (error) code
exch	exchange
fcb	file control block
fh	file handle
lfa	logical file address
rq	request block
ucb	user control block

Suffixes

The suffix identifies the use of the variable (as listed in table 4-3).

Table 4-3. Variable Suffixes

Last	the largest allowable index of an array
Max	the maximum length of an array or buffer (one greater than the largest allowable index)
Ret	the variable's value is to be set by the called process or procedure (it is not specified by the calling process)

Variable Name Examples

Some examples of variable names are:

cbFileSpec	the count of bytes of a file specification
ercRet	the status code to be returned to the calling process
pbFileSpec	the memory address of a string of bytes containing a file specification
pDataRet	the memory address of an area into which data is to be returned to the calling process
ppDataRet	the memory address of a 4-byte memory area into which the memory address of a data item is to be returned to the calling process
pRq	the memory address of a request block
psDataRet	the memory address of a (2-byte) memory area into which the size of a data item is to be returned
sData	the size (in bytes) of a data area
sDataMax	the maximum size (in bytes) of a data area
ssDataRet	the size of the area into which the size of a data item is to be returned.

FORMS RUN TIME SERVICES

Forms Run Time provides many services, categorized by function in table 4-4 and described in alphabetical order in this section.

Table 4-4. Forms Run-Time Services

Form Services	Field Services	User Input Control
DefaultForm	DefaultField	LockKbd
DisplayForm	GetFieldInfo	
OpenForm	ReadField	
UndisplayForm	SetFieldAttrs	
	UserFillField	
	WriteField	

Run Time Sequence

A typical program proceeds as follows:

1. The program calls OpenForm.
2. The program calls DisplayForm.
3. The program then makes a series of calls to UserFillField to accept input, ReadField to encode this input, and WriteField to display decoded data in a field. This is repeated until the form is complete.
4. The program calls UndisplayForm to remove the form from the screen.

The BASIC program in appendix B calls a form per the above procedure.

Defaultfield

DefaultField restores a field to its default value (as it appears immediately following DisplayForm). If **Show default?** is turned on for the field (specified yes when the field was defined), the default value appears.

The procedural interface is:

DefaultField (pForm, pbFieldName, cbFieldName, index):
ErcType

pForm is a pointer to the work area of an open form.

pbFieldName and cbFieldName describe a character string naming the field to be defaulted. The distinction between uppercase and lowercase is ignored in matching field names.

Index specifies which field of a repeating field is to be defaulted. If the named field does not have the repeating property, index is ignored.

Defaultform

DefaultForm restores a form to its default state (as it appears immediately following DisplayForm) by applying DefaultField to each field.

The procedural interface is:

DefaultForm (pForm): ErcType

pForm is a pointer to the work area of an open form.

Displayform

DisplayForm displays a form at a specified location. The form can be centered horizontally and/or vertically within a frame by specifying 255 for iCol and/or iLine. Each field in the form is defaulted as described in DefaultField.

The procedural interface is:

DisplayForm (pForm, iFrame, iCol, iLine): ErcType

pForm is a pointer to the work area of an open form.

iFrame is a frame number in the Video Control Block. (Refer to the *B 20 Systems Operating System (BTOS) Reference Manual.*)

iCol is a column number within the frame.

iLine is a line number within the frame.

GetFieldInfo

GetFieldInfo returns information about a field (for example, the number of repeating fields). Only information requested using cbFieldInfoMax is returned.

The procedural interface is:

```
GetFieldInfo (pForm, pbFieldName, cbFieldName, index,  
pfieldInfoRet, cbFieldInfoMax): ErcType
```

pForm is a pointer to the work area of an open form.

pbFieldName and cbFieldName describe a character string naming the field to be interrogated. The distinction between uppercase and lowercase is ignored in matching field names.

Index specifies which field of a repeating field is to be interrogated. If the named field does not have the repeating property, index is ignored.

pFieldInfoRet and cbFieldInfoMax describe the memory work area into which the field information is returned. The format of this information is listed in table 4-5.

Table 4-5. cbFieldInfoRet and cbFieldInfoMax Format

Offset	FieldL	Size (bytes)	Description
0	iCol	2	a column number (left edge of the form is column 0)
2	iLine	2	a line number (the top of the form is line 0)
4	cCol	2	the field width in columns
6	fShowDefault	2	TRUE if Show-default? is yes
8	fAutoExit	2	TRUE if Auto-exit? is yes
10	fRepeating	2	TRUE if Repeating? is yes
12	attrSel	2	a 4-bit encoding of the selected character attributes
14	attrUnsel	2	a 4-bit encoding of the unselected character attributes
16	indexFirst	2	the lowest index number for a repeating field
18	indexLast	2	the highest (largest) index number for a repeating field
20	reserved	10	
30	cchDefault	2	the length of the default string defined for this field
32	rgchDefault	cchDefault	the default string defined for this field

LockKbd

LockKbd requires that the user press **CANCEL** before using the keyboard. When a program calls LockKbd, the system emits an audio signal and the keyboard locks; further user input is refused until the **CANCEL** key is pressed. Each time a key other than **CANCEL** is pressed, the system emits an audio signal and does not respond.

The procedural interface is:

LockKbd: ErcType

OpenForm

OpenForm reads information associated with a form from an open file into a work area in memory. Once open, the form work area is self-contained and the file can be closed.

The procedural interface is:

OpenForm (fh, pbFormName, cbFormName, pFormRet, cbMax):
ErcType

fh is an open file handle for the form file (or library file).

pbFormName and cbFormName describe a character string containing the form name of the form to be read. The distinction between uppercase and lowercase is ignored in matching form names.

pFormRet and cbMax describe the memory work area into which the open form is returned. A typical form requires a work area of approximately 2K. To find out how much work area the form requires, check the Status Area of the **FORMS EDITOR** or use the Forms Reporter (**FREPORT** command).

ReadField

ReadField reads data from a field into program memory, encoding the currently displayed text according to the type code supplied. **Show default?** has no affect on Readfield. ReadField returns the default value unless a user entry changes the field's value, whether or not the default is displayed.

The procedural interface is:

```
ReadField (pForm, pbFieldName, cbFieldName, index, pbRet,
cbMax, pcbRet, pType): ErcType
```

pForm is a pointer to the work area of an open form.

pbFieldName and cbFieldName describe a character string naming the field to be read. The distinction between uppercase and lowercase is ignored in matching field names.

Index specifies which field of a repeating field is to be read. If the named field is not repeating, index is ignored.

pbRet and cbMax describe the memory work area into which the data is returned.

pcbRet is the memory address of the word into which the count of bytes read is returned.

pType is the memory address of a type code used in encoding the data.

SetFieldAttrs

SetFieldAttrs sets the field character attributes.

The procedural interface is:

```
SetFieldAttrs (pForm, pbFieldName, cbFieldName, index,
attr): ErcType
```

pForm is a pointer to the work area of an open form.

pbFieldName and cbFieldName describe a character string naming the field to be modified. The distinction between uppercase and lowercase is ignored in matching field names.

Index specifies the field in a repeating field to be modified. If the named field is not repeating, the index is ignored.

attr is a word containing any of the following:

- a binary value in the range 0-15, directly encoding the desired character attributes
- an 8-bit character code of a letter in the range A-P (Refer to the table of character attributes in section 2.)
- the letter U for the unselected attributes of the field, as specified at form definition
- the letter S for the selected attributes of the field, as specified at form definition

UndisplayForm

UndisplayForm removes a form from the screen. The form is replaced with the null character (code 0h). The contents of any fields that were not read are lost.

The procedural interface is:

UndisplayForm (pForm): ErcType

pForm is a pointer to the work area of an open form.

UserFillField

UserFillField allows the user to interactively modify a field's contents. Usually this means data entry into a blank field.

UserFillField highlights the field with the selected character attributes and sets the cursor to a specified character position within the field. UserFillField processes user keystrokes until an unrecognized key is depressed, at which point the field is exited and its unselected character attributes are restored. If auto-exit is turned on for a field, that field is exited when the last character cell is filled.

The terminating keystroke is passed back to the calling program in exitState.ch. Usually this program interprets the keystroke (for example, the user may use **NEXT** to sequence through fields). This interpretation is entirely under the control of the program.

A terminating key, such as **NEXT** or **TAB**, is not entered into the field as data. If an auto-exit field is exited because a character is typed in its last position, then the character is entered as data, exitState.ch is set to right arrow (code 12h), and exitState.fAuto-Exit is set to TRUE.

If the user attempts to move the cursor or enter characters beyond the field boundaries, the system emits an audio signal. If the user presses the **CODE** and **Left Arrow** keys, the cursor moves to the left edge of the field; if the user presses the **CODE** and **Right Arrow** keys, the cursor moves to the end of the field.

The user enters or edits text in either insert or overwrite mode and can press the **CODE** and **DELETE** keys or the **CODE** and **BACK SPACE** keys to delete the contents of the entire field.

The procedural interface is:

```
UserFillField (pForm, pbFieldName, cbFieldName, index,  
pInitState, pExitStateRet): ErcType
```

pForm is a pointer to the work area of an open form.

pbFieldName and cbFieldName describe a character string naming the field to be filled. The distinction between uppercase and lowercase is ignored in matching field names.

Index specifies the field that is to be filled in (for a repeating field). If the field is not repeating, the index is ignored.

pInitState points to a descriptor block and pExitStateRet points to a memory area as described in table 4-6.

Table 4-6. pInitState and pExitStateRet Information

Offset	Field	Size (bytes)	Description
pInitState:			
0	ich	2	the initial cursor position relative to the start of the field (where the first character position is numbered 0)
2	reserved	6	
pExitStateRet:			
0	ich	2	the final cursor position relative to the start of the field
2	ch	2	the terminating character code
4	fAutoExit	2	TRUE if the field was exited by auto-exit
6	fModified	2	TRUE if the field contents were changed by the user
8	fEmpty	2	TRUE if the field was empty (all spaces) on exit
10	reserved	6	

WriteField

WriteField writes data to a field from the program memory, decoding the data into text according to the type code supplied.

The procedural interface is:

```
WriteField (pForm, pbFieldName, cbFieldName, index, pb, cb,
pType): ErcType
```

pForm is a pointer to the work area of an open form.

pbFieldName and cbFieldName describe a character string naming the field to be written. The distinction between uppercase and lowercase is ignored in matching field names.

Index specifies which field of a repeating field is to be written. If the named field is not repeating, the index is ignored.

pb and cb describe a memory area containing the data to be displayed.

pType is the memory address of a type code used in decoding the data.

ERROR RESPONSE

Users occasionally make errors while filling in forms. Common errors are alphabetic characters in numeric fields or numeric values that are out of range.

Forms Run Time supports error handling by the program, but it does not mandate a particular format for error response. Two possible ways are:

- You can set the program to signal an error by emitting an audio signal and repositioning the cursor in the field in which invalid data was entered.

The interface to UserFillField permits the

specification of the character position in the field where the cursor is initially positioned; you can use this to respond to an erroneous entry.

This is particularly effective if your program is intended for a user who is typing slowly and looking at the display.

- If the form was designed to contain a special application status field, you can write the program to display an appropriate error message in this field, call LockKbd, and then reset the field to null after the user presses **CANCEL**.

LockKbd emits an audio signal for each character typed, discarding any typing after the erroneous entry until the user presses the **CANCEL** key. You can assign the application status field a character attribute (using the **FORMS EDITOR**) to ensure that any error message is emphasized.

If your program is intended for a user who is doing high-speed data entry and who does not usually look at the screen, this may be necessary to ensure the user realizes an error occurred.

TYPE CODES

A type code is presented to Forms Run Time whenever a field is read or written; the type code determines how the data is encoded or decoded.

User entry data must be encoded (converted) for use by the calling program; data written to a field must be decoded for the display. The encoding and decoding process uses a predefined set of data types. In Forms Run Time, these types are represented by text strings called type codes.

A type code ends with a period and can contain a numeric value (for example, a length). If a type code contains a numeric value, the final period is preceded by a colon and a string of decimal digits. Examples of type codes are Binary and Character:50.

The type system is configurable when Forms Run Time and the program are linked through the use of program-supplied type codes and conversion routines.

CONFIGURING THE TYPE SYSTEM

Type codes obtain their semantics through a table lookup strategy at run time. The table to be searched is determined when you link the Forms Run Time and the program; its name is `rgtdForm` and it is found in the module `FmRgtd.asm`.

The table is generated by assembly language source code. You can extend the type system by changing this table; however, you do not need to use this part of the **FORMS** utility for most applications.

Forms Run Time Structure

The components of Forms Run Time are:

- a master table contained in an assembly language source file, `FmRgtd.asm`
- a set of object module procedures

The result of assembling `FmRgtd.asm` is an object file: `FmRgtd.obj`. The **LINKER** links this object file with the other object modules. (Refer to the *B 20 Systems Linker/Librarian Reference Manual*.)

To configure Forms Run Time, edit `FmRgtd.asm` and add or delete object modules. Then reassemble `FmRgtd.asm` and relink.

FmRgtd.asm Source Text

The source text of FmRgtd.asm is:

```
DGroup GROUP CONST ASSUME DS: DGroup

CONST SEGMENT WORD PUBLIC 'CONST'
%*DEFINE (DefineType(typecode, EncodeProc,
    DecodeProc))
(    DB %LEN(%typecode), '%typecode'
    DD %EncodeProc
    DD %DecodeProc
CONST ENDS
    EXTRN %EncodeProc:FAR, %DecodeProc:FAR
CONST SEGMENT
)

PUBLIC rgtdForm
rgtdForm LABEL BYTE
    %DefineType(Binary, EncodeBinary,
        DecodeBinary)
    %DefineType(Character, EncodeChar,
        DecodeChar)
    DB 0

CONST ENDS

END
```


Defining Types

Each call on the macro DefineType generates one table entry, defining one type. The entry for Binary:

```
%DefineType(Binary, EncodeBinary, DecodeBinary)
```

is equivalent to the following code:

```
        DB 9, 'Binary'  
        DD EncodeBinary  
        DD DecodeBinary  
CONST ENDS  
        EXTRN EncodeBinary:FAR,  
            DecodeBinary:FAR  
CONST SEGMENT
```

This defines the type code Binary, to be implemented by the external procedures EncodeBinary and DecodeBinary.

Starter Kit of Types

All of the encode and decode procedures for these types are implemented by various object modules supplied along with the FORMS utility. Table 4-7 lists the modules that contain conversion procedures.

Table 4-7. Conversion Procedure Modules

Module	Procedure
FmBnry.obj	DecodeBinary
FmChar.obj	EncodeChar, DecodeChar

This set of types can be used as a starter kit. You can add additional types to the list or change the definition of existing types, thereby customizing Forms Run Time to support any specialized data representation or data validation required in a set of applications. Types can be removed from the type system, and the run time thereby made more compact, by deleting the appropriate definition lines from FmRgtd.asm and reassembling. The corresponding object module(s) can then be deleted from the LINKER without error.

For example, to eliminate character strings, the rgtd table should be edited so that it reads:

```
...
rgtdForm LABEL BYTE
           %DefineType(Binary, EncodeBinary,
           DecodeBinary)
           DB 0
...
```

and the module FmReal.obj should be excluded from the LINKER.

When you edit the rgtd table, note that the DB 0 at the end of the table serves as a terminator and must never be deleted.

Adding Types

When type conversion is required by a ReadField or WriteField, the rgtd table is searched for a type code matching the one supplied to the ReadField or WriteField in progress. The distinction between uppercase and lowercase is ignored in matching type codes, as is the numeric suffix of the type code (if any). Thus the type Character, defined above, matches all of the following types codes:

```
CHARACTER:50.
Character:100.
cHarAcTer:1.
```

If no match is found for a given type code, the message **Bad type specification** is returned from ReadField or WriteField. If a match is found in the table, the associated EncodeProc (in the case of ReadField) or DecodeProc (in the case of WriteField) is then called to perform the conversion.

An EncodeProc (used by ReadField) must have the interface:

```
EncodeProc (prgch, cch, pb, cb, pcbRet, sType): ErcType
```

prgch is a pointer to the text read by Forms Run Time from a field.

cch is the length of the text read from a field.

pb and cb describe the data area passed to ReadField where the encoding of the text should be placed.

pcbRet is a pointer to a word to be set to the number of encoded bytes placed by EncodeProc into pb.

sType is the binary value of the type code's numeric suffix (if any).

EncodeProc converts the string defined by prgch and cch into data in the area defined by pb and cb and sets the word pointed to by pcbRet to the number of bytes returned. sType gives specific information about the data types (needed to conveniently handle types, such as packed decimal). If encoding is unsuccessful or a data validation fails (for example, because an alphabetic character is entered into a numeric field), the message **Invalid data** is returned; otherwise, the message **Ok** is returned.

DecodeProc (used by WriteField) must have the interface:

```
DecodeProc (prgch, cch, pb, cb, pcbRet, sType): ErcType
```

prgch is a pointer to an area in which text to be written to a field is placed.

cch is the length of an area in which text to be written to a field is placed.

pb and cb describe the existing encoded data.

pcbRet is a pointer to a word to be set to the number of decoded bytes.

sType is the binary value of the type code's numeric suffix (if any).

Example:

To implement a new type called Money, proceed as follows:

1. Write procedures EncodeMoney and DecodeMoney to perform the conversions. EncodeMoney accepts \$ddd.cc strings, converts them to cents, and then converts them to 16-bit integers equal to the number of cents. If a string containing other characters or in a different format were read, EncodeMoney displays the message **Invalid data**. DecodeMoney converts a binary number of cents into a money string with \$ and . punctuation.
2. Edit the FmRgtd.asm file to include a type definition for Money, by adding an entry of the form:

%DefineType(Money, EncodeMoney, DecodeMoney)
3. Relink the reassembled FmRgtd.obj to the application system along with modules containing the procedures EncodeMoney and DecodeMoney.

The system then recognizes the type code Money and Forms Run Time calls EncodeMoney and DecodeMoney whenever needed.

Range Checking and Data Validation

The program is responsible for range checking and other data validation of encoded data. In the BASIC program in appendix B, the PartNumber field is checked for range validity in lines 720-740.

APPENDIX A

ERROR MESSAGES

The **FORMS** utility has status and error messages to assist you in creating a form or calling the form from a program.

FORMS EDITOR ERROR MESSAGES

Bad file format

The file typed in a **READ FORM** command form cannot be found. Check that you have specified the file correctly and try again. The volume and directory default is the current path. If you want to access a form from a different volume or directory, you must type in the complete file name using brackets--[volume]<directory>filename.

Cannot move partial fields

You can only **MOVE** or **COPY** a partial field to a location on the same line. Adjust your selection to include the entire field or **MOVE** or **COPY** the partial field to a location on the same line.

Cannot open that file

The file typed in a **READ FORM** command form cannot be found. Check your file specification. The volume and directory default is the current path. If you want to access a form in a different volume or directory, you must type in the complete file name using brackets--[volume]<directory>filename.

Destination out of bounds

You cannot **MOVE** or **COPY** a field into an area which overlaps the edge of the screen. Check the position of your selection and check the position of the cursor.

Field has nonsequential index

One field of a repeating field has an index that is out of sequence with the rest of the repeating field. The field is blinking. Change the index or rename the field.

Form is too complex

Your form cannot be translated into binary format because it has exceeded the **FORMS EDITOR** in-memory work area. Try a **TEST DRIVE** (the **FORMS EDITOR** compacts the form as a result of **TEST DRIVE** and **WRITE FORM** commands). If the message still displays, reduce the length of field names or default values until the message no longer displays when you **TEST DRIVE** the form. This problem is particularly likely to occur if you moved or copied fields.

Form is too wide

The **ZOOM** command cannot reformat the display to 80 columns because the form does not fit. Use the **VIEW EDIT CODE** command (press **CODE** and **V**) to check for any space characters that you can remove.

Illegal form name

A form name must consist only of alphanumeric characters (including the filename). Make sure that your entire form and file name conforms to this restriction.

Multiple use of same field name

Two or more fields have the same name, but they are not defined as repeating. The problem fields are blinking. Either change the field names or specify yes for repeating.

No fields defined

NEXT and **TEST DRIVE (f9)** cannot be used unless a field has been defined.

No selection

The command you chose requires a selection. Make a selection and then try the command.

No such form

The form you typed in the **READ FORM** command form does not exist in that file. If your file name is correct, check your current path.

Too many attributes (B 21 workstation only)

B 21 systems have a limit of 15 attribute changes per line. An attribute change occurs once at the beginning of the line and whenever two adjacent characters have different attributes. In addition to character attributes discussed in section 2, B 21 systems consider changes between ASCII code ranges 0 through 127 (7Fh) and 128 (80h) through 255 (OFFh) as character attribute changes.

If you exceeded the character attribute limitation on a B 21 because you have 7 or more vertical lines on one line, you can use **CODE '** to create the vertical lines (character code 16h).

Too many fields

Your form cannot be translated into binary format because it has exceeded the **FORMS EDITOR** in-memory work area. Try a **TEST DRIVE** (the **FORMS EDITOR** compacts the form as a result of **TEST DRIVE** and **WRITE FORM** commands). If the message still displays, reduce the length of field names or default values until the message no longer displays when you **TEST DRIVE** the form. This problem is particularly likely to occur if you moved or copied fields.

FORMS RUN TIME ERROR MESSAGES

<u>Decimal Value</u>	<u>Hexa-decimal Value</u>	
3700	0E74	Name not found. The form name supplied to OpenForm was not found in the file. Check that the file name and form name are correct for the form you want.
3701	0E75	Bad object file. The file supplied to OpenForm is not a valid object module. The file could be empty. Check that the file name is correct for the form you want.
3702	0E76	Form too big. The work area supplied to OpenForm is too small to contain the named form. You can use the Forms Reporter (FREPORT command) to determine the required work area.
3703	0E77	Form out of bounds. The screen coordinates passed to DisplayForm would result in a part of the form lying outside the frame. You can use the Forms Reporter (FREPORT command) to determine the required height and width.

<u>Decimal Value</u>	<u>Hexa-decimal Value</u>	
3704	0E78	<p>Form not displayed.</p> <p>A Forms Run Time service (DefaultField, DefaultForm, ReadField, SetFieldAttrs, UndisplayForm, UserFillField, or WriteField) was called for a form that was not displayed. Display the form (use DisplayForm) before attempting any of these Forms Run Time services.</p>
3705	0E79	<p>No such field.</p> <p>A Forms Run Time service (DefaultField, GetFieldInfo, ReadField, SetFieldAttrs, UserFillField, or WriteField) was called for a field (specified by pbFieldName, cbFieldName) and index that does not exist. Use the Forms Reporter (FREPOR command) to determine the correct name and index.</p>
3706	0E7A	<p>Bad type specification.</p> <p>A ReadField or WriteField was supplied with a type code that is not defined in your configuration. Examine the source text of FmRgtd.asm for a list of defined type codes.</p>
3707	0E7B	<p>Bad data size.</p> <p>A ReadField was attempted in which the cbMax parameter was incorrect for the type of data being returned (for example, a cbMax of three for type Binary). Make sure that the size and type of your data agree.</p>

<u>Decimal Value</u>	<u>Hexa- decimal Value</u>
----------------------	----------------------------

3708	0E7C	Invalid data.
------	------	---------------

A ReadField or WriteField was attempted in which the requested data conversion could not be performed (for example, reading an alphabetic string as type Binary). For WriteField, make sure the type of the data you are displaying is correct. For ReadField, display an error message and have the user reenter the data.

APPENDIX B

SAMPLE BASIC PROGRAM

This BASIC program uses OpenForm, DisplayForm, UserFillField, ReadField, WriteField, and UndisplayForm (Forms Run Time services discussed in section 4) to run the form created in appendix C. The program includes range checking and data validation of encoded data with the PartNumber field (lines 720-740).

```
10      'Basic program using Tutorial.form
11      'Illustrating fixed and repeating fields, user input and
        program output
12      OPTION BASE 1
13      CHNEXT% = &HA: ERCINVALIDDATA% = 3708
14      'Create part number <=> price data base
15      DIM PRICES% [20]
16      FOR I% = 1 TO 20: READ PRICES%[I%]: NEXT I%
17      DATA 1, 5, 17, 20, 3, 4, 7, 6, 2, 3, 3, 12, 15, 19, 6, 8,
        4, 9, 9, 15
18
19      'Open the form
20      FILE$ = "Tutorial.form"
21      PSWD$ = ""
22      FH% = 0
23      MODE% = &H6D72 ' modeRead
24      ERC% = OPENFILE(PTR(FH%), PTR(FILE$), LEN(FILE$),
        PTR(PSWD$), LEN(PSWD$), MODE%)
25      IF ERC% <> 0 THEN GOTO 9000
26      FORM$ = "Tutorial"
27      DIM FORM% [500] ' 500 words = 1000 bytes
28      ERC% = OPENFORM(FH%, PTR(FORM$), LEN(FORM$),
        PTR(FORM%[1]), 1000)
29      ERC2% = CLOSEFILE(FH%) ' close the file even if OpenForm
        got an error
30      IF ERC% <> 0 THEN GOTO 9000
31      IF ERC2% <> 0 THEN GOTO 9000
32      'Now clear the screen and display the form centered
33      PRINT CHR$(255) + "pf" + CHR$(255) + "vf" + CHR$(&HC);
```

```

250   ERC% = DISPLAYFORM(PTR(FORM%[1]), 0, 255, 255)
260   IF ERC% <> 0 THEN GOTO 9000
270   '
300   'Prompt user to fill in Salesman field
310   DIM INITSTATE% [4]
320   DIM EXITSTATE% [8]
330   FLD$ = "Salesman"
340   INITSTATE%[1] = 0 ' initState.ich: initial cursor
      position
350   ERC% = USERFILLFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
      0, PTR(INITSTATE%[1]), PTR(EXITSTATE%[1]))
360   IF ERC% <> 0 THEN GOTO 9000
370   IF EXITSTATE%[2] = CHNEXT% THEN GOTO 500 '
      exitState.ch = Next => proper exit
380   PRINT CHR$(7); ' anything else is improper exit: make a
      beep (by printing ASCII bell)
390   INITSTATE%[1] = EXITSTATE%[1] ' leave cursor where it was
400   GOTO 350 ' and try again
410   '
500   'Now find out how many repetitions there are
510   DIM INFO% [16]
520   FLD$ = "PartNumber"
530   ERC% = GETFIELDINFO(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
      1, PTR(INFO%[1]), 32)
540   IF ERC% <> 0 THEN GOTO 9000
550   '
560   'Loop through all repetitions
570   AMOUNTDUE% = 0
580   FOR INDEX% = INFO%[9] TO INFO%[10] ' indexFirst to
      indexLast
590   '
600   'Prompt user to fill in Part Number
610   PART% = 0
620   FLD$ = "PartNumber"
630   Type$ = "Binary."
640   INITSTATE%[1] = 0 ' initial cursor position
650   ERC% = USERFILLFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
      INDEX%, PTR(INITSTATE%[1]), PTR(EXITSTATE%[1]))
660   IF ERC% <> 0 THEN GOTO 9000
670   IF EXITSTATE%[2] = CHNEXT% THEN GOTO 710 ' proper exit
680   PRINT CHR$(7); ' improper exit: beep
690   INITSTATE%[1] = EXITSTATE%[1] ' leave cursor where it was
700   GOTO 650 ' and try again
710   'Find out what the user typed
715   CBREAD% = 0

```

```

720     ERC% = READFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
INDEX%, PTR(PART%), 2, PTR(CBREAD%), PTR(TYPE$))
730     IF ERC% = 0 AND PART% > 0 AND PART% <= 20 THEN GOTO 800 '
valid part number
740     IF ERC% <> ERCINVALIDDATA% AND ERC <> 0 THEN GOTO 9000 '
random error
750     'Here if invalid part number:  display an error message
760     MSG$ = "Invalid Part Number"
770     GOSUB 5000
780     ' And make the user do it all over again
790     GOTO 600
800     GOSUB 6000 ' clear the message
810     'Got a good part number, redisplay to right justify
820     ERC% = WRITEFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
INDEX%, PTR(PART%), 2, PTR(TYPE$))
825     IF ERC% <> 0 THEN GOTO 9000
830     'Now look up unit price and display
840     UNITPRICE% = PRICES%[PART%]
850     FLD$ = "UnitPrice"
860     TYPE$ = "Binary."
870     ERC% = WRITEFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
INDEX%, PTR(UNITPRICE%), 2, PTR(TYPE$))
880     IF ERC% <> 0 THEN GOTO 9000
890     '
900     'Prompt user to fill in Quantity
910     FLD$ = "Quantity"
920     TYPE$ = "Binary."
930     QUANTITY% = 0
940     INITSTATE[1] = 0 ' initial cursor position
950     ERC% = USERFILLFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
INDEX%, PTR(INITSTATE[1]), PTR(EXITSTATE[1]))
960     IF ERC% <> 0 THEN GOTO 9000
970     IF EXITSTATE[2] = CHNEXT% THEN GOTO 1010 ' proper exit
980     PRINT CHR$(7); ' improper exit:  beep
990     INITSTATE[1] = EXITSTATE[1] ' leave cursor where it was
1000    GOTO 950 ' and try again
1010    'Find out what the user typed
1015    CBREAD% = 0
1020    ERC% = READFIELD(PTR(FORM%[1]) PTR(FLD$), LEN(FLD$),
INDEX%, PTR(QUANTITY%), 2, PTR(CBREAD%), PTR(TYPE$))
1030    IF ERC% = 0 THEN GOTO 1100 ' valid number
1040    IF ERC% <> ERCINVALIDDATA% THEN GOTO 9000 ' random error
1050    'Here if invalid quantity:  display an error message
1060    MSG$ = "Not a number"
1070    GOSUB 5000

```

```

1080 'And make the user do it all over again
1090 GOTO 900
1100 GOSUB 6000 ' clear the message
1110 'Redisplay quantity as we did part number
1120 ERC% = WRITEFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
INDEX%, PTR(QUANTITY%), 2, PTR(TYPE$))
1125 IF ERC% <> 0 THEN GOTO 9000
1130 'Compute total price = unit price * quantity, and display
1140 TOTALPRICE% = UNITPRICE% * QUANTITY%
1150 FLD$ = "TotalPrice"
1160 TYPE$ = "Binary."
1170 ERC% = WRITEFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
INDEX%, PTR(TOTALPRICE%), 2, PTR(TYPE$))
1180 IF ERC% <> 0 THEN GOTO 9000
1190 'And update Amount Due
1200 AMOUNTDUE% = AMOUNTDUE% + TOTALPRICE%
1210 FLD$ = "AmountDue"
1220 TYPE$ = "Binary."
1230 ERC% = WRITEFIELD(PTR(FORM%[1]), PTR(FLD$), LEN(FLD$),
INDEX%, PTR(AMOUNTDUE%), 2, PTR(TYPE$))
1240 IF ERC% <> 0 THEN GOTO 9000
1250 '
1400 'Finished a row, loop back for next row
1410 NEXT INDEX%
1420 '
1500 'Last row filled: put cursor back and exit
1510 PRINT CHR$(255) + "pn" + CHR$(255) = "vn";
1520 END
1530 '
5000 'Subroutine to display MSG$ in the form
5010 'Use escape sequence to position cursor and set blinking
attributes
5020 PRINT CHR$(255) + "c" + CHR$(50) + CHR$(25) + CHR$(255) +
"ai";
5030 'Display message
5040 PRINT MSG$;
5050 'Reset attributes to plain and beep
5060 PRINT CHR$(255) + "aa" + CHR$(7);
5070 RETURN
5080 '
6000 'Subroutine to clear the error message
6010 'Use escape sequence to position cursor
6020 PRINT CHR$(255) + "c" + CHR$(50) + CHR$(25);
6030 'Print 50 spaces
6040 PRINT STRING$(50, " ");

```

```
6050 RETURN
6060 '
9000 'Error exit
9010 PRINT "error ", ERC%
9020 GOTO 1500
```


APPENDIX C

TRAINING EXERCISE: CREATE A FORM

This appendix is a training exercise containing step-by-step instructions for creating a form (with captions, lines and boxes, and single and repeating fields).

If you are not familiar with overtype and insert modes or cursor movement on the B 20, you should review those paragraphs in section 2. Also refer to section 2 as necessary for additional information on **FORMS EDITOR** commands.

NOTE

You can exit any **FORMS EDITOR** command (**TEST DRIVE**, **WRITE FORM**, etc.) and return to the **FORMS EDITOR** by pressing **CANCEL**. You can exit the **FORMS EDITOR** and return to the Executive by pressing **FINISH**.

ACCESS THE FORMS EDITOR TO CREATE A FORM

To access the **FORMS EDITOR**, proceed as follows:

1. Type *forms editor* at a command prompt (B 20 Executive).
2. Press **GO**. The words **Forms Editor** appear at the top of the screen. The area below the double line is work space for creating a form. The cursor is in the center of the screen.


START WITH CAPTIONS

1. Move the cursor approximately 16 spaces to the left.
2. Type *Part No.*, then press the **Right Arrow** key three times.
3. Type *Quan.*, then press the **Right Arrow** key three times.
4. Type *Price*, then press the **Right Arrow** key three times.
5. Type *Total*. The center of your form should now look similar to :

Part No.	Quan.	Price	Total
----------	-------	-------	-------

ADD SOME LINES AND BOXES

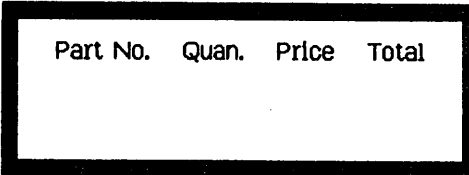
1. Surround the captions with a box. Proceed as follows:
 - 1) Position the cursor one line above and two character cells to the left of the **Part No.** caption. Press **MARK**.
 - 2) Position the cursor three lines below and two character cells to the right of the **Total** caption. Press **BOUND**. You have now selected an area for a box. Your form should now look similar to figure C-1.



Part No.	Quan.	Price	Total
----------	-------	-------	-------

Figure C-1. Selected Caption Box

- 3) Press the **SHIFT** and **f4** keys (**DRAW**). The captions are surrounded by a thick line box. Your form should now look similar to figure C-2.



Part No.	Quan.	Price	Total
----------	-------	-------	-------

Figure C-2. Captions In A Box

2. Subdivide the box horizontally (draw lines to separate the captions). Proceed as follows:
 - 1) Position the cursor one line below the captions at the right box boundary. Press **MARK**.
 - 2) Position the cursor on the same line (one line below the captions) at the left box boundary. Press **BOUND**.

Since you have selected an area one character cell high, your selection is for a line. Your form should now look similar to figure C-3.

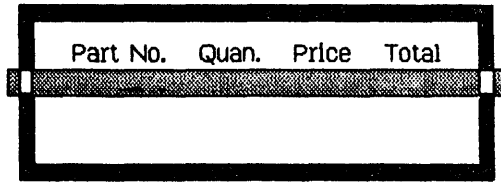


Figure C-3. Selected Horizontal Line

- 3) Press **f4 (DRAW)**. A thin line appears.

3. Subdivide the box vertically. Proceed as follows:

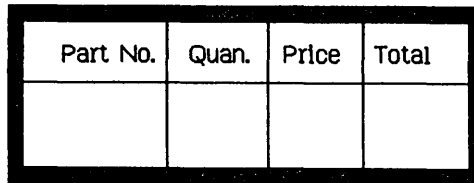
- 1) Midway between **Part No.** and **Quan.**, select a rectangle one character cell wide which intersects the top and bottom thick lines.

(Position the cursor one line above the captions, press **MARK**. Position the cursor in a direct line below the selected character cell, intersecting the bottom line. Press **BOUND**.)

- 2) Press **f4 (DRAW)**.
- 3) Add vertical lines between **Quan.** and **Price** and between **Price** and **Total**.

(Press **f1 (RESELECT)**.) The selection for the line between **Part No.** and **Quan.** reappears. Position the cursor one line above the captions, midway between **Quan.** and **Price**. Press **COPY** twice. Two more vertical lines appear. (You can use **COPY** because these columns are equal in area.)

Your form should now look similar to figure C-4.



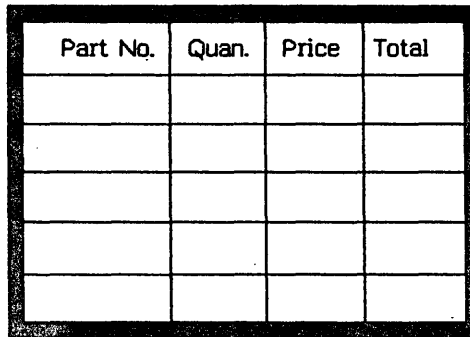
Part No.	Quan.	Price	Total

Figure C-4. Subdivided Box

ADD TABULAR COLUMNS

To make a tabular form with five rows of data, you **COPY** the lower box five times. Proceed as follows:

1. Select the lower box using **MARK** and **BOUND**.
(Position the cursor on the left boundary, one line below the caption. Press **MARK**. Position the cursor on the right boundary, two lines below the marked corner. Press **BOUND**.)
2. Position the cursor one line below the lower left corner of the box. The cursor is now in the lower left corner of the selection.
3. Press **COPY**. A copy of the selected box is appended to the bottom of the box and selected. The cursor is positioned one line below the lower left corner of the box (in the lower left corner of the selection).
4. Press **COPY** three more times. Your form should now look similar to figure C-5.



Part No.	Quan.	Price	Total

Figure C-5. Tabular Form

DEFINE FIELDS FOR USER ENTRY

1. Define repeating fields for each column. Figure C-6 illustrates the screen during the define field command form with the part number repeating field selected. Proceed as follows:
 - 1) Select the column labeled **Part No.** as shown in figure C-6. (Be careful not to include the caption line in your selection.)

(Move the cursor to inside the left bottom corner of the form. Press **MARK**. Position the cursor one line below and left of the intersection between the **Part No.** and **Quan.** columns. Press **BOUND**.)
 - 2) Press **f8 (DEFINE FIELD)**. The **DEFINE FIELD** command form appears. Your screen should now look similar to figure C-6.

Use **NEXT** to move the cursor through the **DEFINE FIELD** command form fields, as explained in the following steps. When changing an existing entry (such as the **Yes** for **Show default?**), you must use **DELETE** to clear the character cells before before you can type a new entry.
 - 3) Type *PartNumber* in the **Name:** field of the **DEFINE FIELD** command form.
 - 4) Press **GO**. Five fields are defined. Each contains a tag (sequence of square boxes).
 - 5) Select the **Quan.** column and define a repeating field named quantity with a default value of 1, and set **Show Default?** to no.
 - 6) Define the repeating fields **UnitPrice** and **TotalPrice**. Your form should now look similar to figure C-7.

DEFINE FIELD

Name: Index:

Default value:

Options:
Show default? Auto-exit? Repeating?

Attributes: Unselected: Selected:

Part No.	Quan.	Price	Total
<input type="text"/>			

Figure C-6. Selection For A Repeating Field

Part No.	Quan.	Price	Total
□□□□□□□□	□□□□□□	□□□□□□	□□□□□□
□□□□□□□□	□□□□□□	□□□□□□	□□□□□□
□□□□□□□□	□□□□□□	□□□□□□	□□□□□□
□□□□□□□□	□□□□□□	□□□□□□	□□□□□□
□□□□□□□□	□□□□□□	□□□□□□	□□□□□□

Figure C-7. Defined Fields

ADDITIONAL TRAINING ACTIVITIES

Before exiting the **FORMS EDITOR**:

- Save the form you created in a file named Tutorial by using the **WRITE FORM** command (press **f7**).
- Run a **TEST DRIVE** (press **f9**) of the form you created.

After exiting the **FORMS EDITOR** (exit by pressing **FINISH**):

- Run a **FORMS REPORTER** report on the form you created (command **FREPORT** at the Executive level).
- Use the **BASIC** program in appendix B to run your form.

APPENDIX D

GLOSSARY OF TERMS

CHARACTER ATTRIBUTES

Character attributes are visual highlights which you can apply to fields when you define them. These highlights make the defined field blink, display in reverse video, display half-bright, and/or display underlined.

DEFAULTFIELD

DefaultField is a Forms Run Time service that restores a field's value to its default.

DEFAULTFORM

DefaultForm is a Forms Run Time service that restores a form to its default state.

DEFINE FIELD

DEFINE FIELD (f8) is a FORMS EDITOR command that allows you to set field characteristics (such as field name, default, character attributes).

DELETE SELECTION

DELETE SELECTION (f10) is a FORMS EDITOR command that removes all captions, lines, and fields from a selected area.

DISPLAYFORM

DisplayForm is a Forms Run Time service that displays a form on the screen.

DRAW

DRAW (f4) is a **FORMS EDITOR** command that allows you to draw a line or box at the selected area of the form.

ERASE

ERASE (f5) is a **FORMS EDITOR** command that allows you to remove a line or box from the selected area of the form.

FIELDS

Fields are special areas of a form that are set aside for accepting user data entries or displaying computed data. A form can contain many nonoverlapping fields, limited by screen size and byte availability of the system.

FORMS

Forms are lined and captioned displays with fields for accepting user data entries or displaying computed data.

FORMS REPORTER

The Forms Reporter (command **FREPORT**) displays the following form information: the form name, the size (bytes), the displayed height and width, and information on the defined fields.

FORMS RUN TIME

Forms Run Time is a library of object module procedures (services).

GETFIELDINFO

GetFieldInfo is a Forms Run Time service that returns information about a field.

INDEX

Index numbers are used to distinguish locations in a repeating field.

INSERT MODE

Insert is one of the edit modes available on the B 20. In insert mode, the characters you type are inserted at the current cursor position (the character at the cursor position and the cursor move to the right). If the **OVER TYPE** key light is not on, your B 20 is in insert mode. If the light is on, press the **OVER TYPE** key to enter the insert mode.

LOCKKBD

LockKbd is a Forms Run Time service that requires the user to press **CANCEL** before using the keyboard.

OPENFORM

OpenForm is a Forms Run Time service that reads a form from a file into a work area in memory.

OVERTYPE MODE

Overtime is one of the edit modes available on the B 20. In overtime mode the characters you type replace the characters at the cursor position. If the **OVER TYPE** key light is on, your B 20 is in overtime mode. If the light is not on, press the **OVER TYPE** key to enter the overtime mode.

READ FORM

READ FORM (f6) is a **FORMS EDITOR** command that allows you to display a stored form.

READFIELD

ReadField is a Forms Run Time service that reads data from a field into program memory.

REPEATING FIELDS

A repeating field is a group of fields that have the same field name. The repeating fields are distinguished by their indexes (location).

RESELECT

RESELECT (f1) is a FORMS EDITOR command that allows you to select the same form area again.

SELECTIONS

A selection is a rectangular area of the screen that you choose by using the MARK and BOUND keys.

SETFIELDATTRS

SetFieldAttrs is a Forms Run Time service that sets the character attributes of a field on the screen.

TEST DRIVE

TEST DRIVE (f9) is a FORMS EDITOR command that allows you to display the form as it will appear at run time and fill in fields as a user.

UNDISPLAYFORM

UndisplayForm is a Forms Run Time service that removes a form from the screen.

UNDO

UNDO (f2) is a **FORMS EDITOR** command that allows you to delete the previous change to the form.

USERFILLFIELD

UserFillField is a Forms Run Time service that allows user input to a field.

WRITE FORM

WRITE FORM (f7) is a **FORMS EDITOR** command that allows you to store (save) a form.

WRITEFIELD

WriteField is a Forms Run Time service that writes data to a field from program memory.

INDEX

- Adding Types 4-19
- Auto-exit 1-4
- Back Space Key 2-7
- Bound 2-5
- Captions 2-6
- cbFieldInfoMax 4-8
- cbFieldInfoRet 4-8
- Character Attributes 2-15
- Codes, Type 4-15
- Configuring the Type System 4-16
- Conversion Procedure Modules 4-18
- Copy 2-20
- Cursor 2-4
- Data Validation 4-22
- Default 1-4
- DefaultField 4-5
- DefaultForm 4-6
- Define Field (f8) 2-13
- Defining Repeating Fields 2-16
- Defining Single Fields 2-13
- Defining Types 4-18
- Delete Key 2-7
- Delete Selection (f10) 2-19
- DisplayForm 4-6
- Draw (f4) 2-9
- Edit Modes 2-6
- Erase (f5) 2-10
- Error Messages
 - Forms Editor A-1
 - Forms Run Time A-4
- Error Response (Forms Run Time) 4-14
- Exiting the Forms Editor 2-23
- Field Definitions 1-3
 - Auto-exit During Program Run 1-4
 - Default 1-4
 - Highlights 1-4
 - Modifying Field Definitions 2-18
 - Modifying Field Position And Width 2-18
 - Name 1-3
 - Repeating Fields 1-3

INDEX (Cont)

Fields 1-2
Finish 2-23
FmRgtd.asm Source Text 4-17
Forms 1-2
Forms Editor 1-2, 2-1
Forms Editor Commands 2-8
Forms Reporter 1-4, 3-1
Forms Run Time 1-5, 4-1
Forms Run Time Services 4-4
Forms Run Time Structure 4-16
Freport 1-4, 3-1
GetFieldInfo 4-7
Highlights 1-4
Insert 2-7
Installing The B 20 Forms Utility 1-1
Linker/Librarian 1-2, 1-6, 2-12, 4-16, 4-19
Literal Insert 2-7
LockKbd 4-9
Mark 2-5
Move 2-21
Next Tag 2-22
OpenForm 1-6, 4-9
Opening a Form 1-6
Overtyping 2-6
pExitStateRet 4-13
pInitState 4-13
Prefixes 4-2
Range Checking and Data Validation 4-22
Read Form (f6) 2-10
ReadField 4-10
Repeating Fields 1-3
Reselect (f1) 2-8
Roots 4-3
Run Time Sequence 4-5
Save Form 2-23
Selections 2-5
SetFieldAttrs 4-10
Show Default 1-4, 4-5, 4-10
Software Installation 1-1
Starter Kit of Types 4-18
Status and Work Areas 2-3
Suffixes 4-3
Tag 2-15
Test Drive (f9) 2-19

INDEX (Cont)

- Tutorial 3-3, C-11
- Type Codes 4-15
 - Adding Types 4-19
 - Configuring the Type System 4-16
 - Data Validation 4-22
 - Defining Types 4-18
 - FmRgtd.asm Source Text 4-17
 - Forms Run Time Structure 4-16
 - Range Checking 4-22
 - Starter Kit of Types 4-18
- UndisplayForm 4-11
- Undo (f2) 2-8
- UserFillField 4-11
- Variable Names 4-1
- Variable Prefixes 4-2
- Variable Root Terms 4-3
- Variable Suffixes 4-3
- View Edit Codes 2-22
- Work Area 2-3
- Write Form (f7) 2-12
- WriteField 4-14
- Zoom (Code Z) 2-22

