UNISYS

BTOS

Forms Designer

Programming Guide

Relative to Release
Level 6.0

Priced Item

October 1987
Distribution Code SA
Printed in U S America
5027212

UNISYS

BTOS
Forms Designer
Programming Guide

Relative to Release Level 6.0	October 1987 Distribution Code SA Printed in U S America
Priced Item	5027212

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material 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.

Comments or suggestions regarding this document should be submitted on a Field Communication Form (FCF) with the CLASS specified as 2 (S.S.W.:System Software), the Type specified as 1 (F.T.R.), and the product specified as the 7-digit form number of the manual (for example, 5027212).

About This Guide

This guide contains reference information and programming procedures for BTOS Forms Designer software, including Forms Editor, Forms Reporter, and Forms Run-Time.

Who Should Use This Guide

This guide is for BTOS Forms Designer programmers. To understand some of the information in this guide, you must be familiar with:

□ standard workstation operating system Executive commands

□ linking a form to a program

How to Use This Guide

If you are using BTOS Forms Designer for the first time, you should read sections 2 and 3. They contain basic information you will need for installing the software and creating, editing, and reporting a form.

In any case, if you scan the contents and review the topics before you start, you may find this manual easier to use. To find definitions of unfamiliar words, use the glossary; to locate specific information, use the index.

How This Guide is Arranged

This material is divided into sections, with related subjects grouped together. Section 1 describes the basic concepts involved in the operations. Thereafter, the general sequence of topics covers installing, using, and customizing the software, and the representation of forms in memory and on a BTOS disk file.

Procedures

Sections 2, 3, and 4 contain procedures for BTOS Forms Designer operations.

For installation, refer to section 2.

For Forms Designer operations, refer to section 3.

For customizing Forms Designer, refer to section 4.

For descriptions of the external and internal representation of a form, refer to section 5.

Conventions

The following conventions apply to all procedures:

- When two keys are used together for an operation, their names are hyphenated (for example, **ACTION-GO**).
- The term character includes spaces.

Example Programs

Appendixes B and C contain complete programs in BASIC and PASCAL languages. Appendix D provides you with a training exercise for creating a form.

Reference Material

This guide contains appendixes with reference information, a glossary, and an index.

For definitions of key terms used in this manual or related to this software, refer to the glossary.

Related Product Information

For detailed information about using the mouse device, refer to the *BTOS II System Reference Manual.*

For more about linking a form to a program, you can refer to *BTOS II Language Development Linker and Librarian Programming Guide.*

For an explanation of workstation operating system Executive Commands, refer to the *BTOS II Standard Software Operations Guide.* For information about BTOS, refer to the *BTOS II System Reference Manual.*

Contents

Illustrations

Tables

Overview

The BTOS Forms Designer allows you to create, revise, and incorporate forms into your programs that request and accept user input.

The Forms Designer software includes:

- a Forms Editor (FORMS EDITOR command) to help you design forms
- a Forms Reporter (FREPORT command) to display information about a form you have created
- Run Time Modules (Forms Run-Time) that you link to a program to display forms and accept data that you supply.

You design or edit forms (draw lines, enter captions, and define fields for user entries) 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 the form as it appears at program run time. The Forms Editor also has a Test Drive function that allows you to fill in fields as a user. When you use the Test Drive function or the Write Form function to save your form, the Forms Editor converts the form into an object module. Refer to section 3 for information.

When you finish creating or editing a form, you may save the form for run time access, add it to a library of similar forms using the BTOS Librarian, or link it directly with your application.

Forms, Fields, and Files

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 display size and byte availability. You define the fields when you create or change a form.

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

Field Definitions

Lines and captions are protected fields; that is, user input at run time cannot change them. You define unprotected fields (the fields for user input) by using the Define Field function of the Forms Editor (refer to section 3).

Defined fields have:

□ field name

□ index (if repeating fields)

□ default value (optional, value that appears in the field)

□ sequence number

□ data type control (Tx, An, Ab, or Nu)

□ justification (L, R, or a number in the range 1-15)

□ mandatory

□ protected

□ secret property

□ show default

□ automatic exit when the user enters a last character

□ repeating

□ selected/unselected character attribute (up to 16 character attribute combinations can be applied to select or unselect parameters. The attributes are listed in table 3-3.

□ user validation routine

□ help message

□ list of values

□ selected/unselected color (0,1,...7)

Field 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. In matching field names, Forms Run-Time ignores the distinction between uppercase and lowercase.

Default Field Value

Each field has a default value that the field is set to when Forms Run-Time displays the form. The default is empty (null) unless you enter a value when you define (or redefine) the field.

Unless you set the **Show Default?** field to **No** when you define a field, the default displays when the form is used. Forms Run-Time reads the default value whether or not it displays.

Sequence Number

The sequence number applies only if TypeForm controls form fill-in processing.

When the user exits a field, TypeForm checks if the exit was caused by a Next field or a Previous field function key, by the auto-exit property (equivalent to a Next field key), or by a First field or a Last field function key.

TypeForm chooses the field to be selected next. The default orders the fields depending on their location on the screen, from top to bottom and from left to right. The user may explicitly define the sequence number of each field. The sequence numbers associated with the different fields of a form must be a continuous range of values starting at 1.

Data Type Control

Data types can be any of the following:

- alphabetic - lowercase and uppercase letters only
- numeric - 0...9, +, –, decimal mark in decimal fields
- alphanumeric - letters, figures and usual special characters
- text - all ASCII characters (default)

When the user or the application program enters a character into a field, Forms Run-Time verifies that the character is the required type for that field. If it is not, and the user has entered an incorrect character, an error message displays. If an application program has entered an incorrect character, an error code is returned to the program.

A space is considered as an alphanumeric character, except when Forms Run-Time is checking whether a field is empty or not. The uninitialized part of a field always contains binary zeroes and not spaces, although binary zeroes are never returned to the application program.

Justification

Justification of text in a field can be:

- left-justified (default)
- right-justified
- justified on the decimal point (the user specifies a fixed number of decimals when defining the field)

Field Type: Mandatory, Protected, or Optional

The field type can be mandatory, protected, or optional.

Mandatory fields are fields that cannot be left empty. An empty field contains no other character than space(s) or binary zero. This is checked:

- each time the user exits a mandatory field that has been selected for being filled in; or
- for all mandatory fields in a form when the user presses a form validation key

Note: **If the field is initialized with a default value or a value written by the application program, the user is not required to modify this value.**

A protected field allows data to be entered by an application program, but not by the user. The TypeForm operation automatically skips over protected fields.

Optional fields (default) include all other fields.

Secret Property

If a field is defined with the secret property, every character entered (by an application program or by the user) displays as # on the screen.

Field Auto-exit During Program Run

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

Repeating Fields

Repeating fields are groups of fields that have the same name. For example, 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 from one another by their indexes (locations). 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 function of the Forms Editor). No two fields (locations) can have the same field name and index number.

You can change 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 procedure to determine the number of fields (refer to section 4).

Character Attributes

You can apply visual character attributes (highlights) to fields when you define them. The character attributes available are:

- blinking
- reverse video
- half-bright
- underlining

You can assign a selected and an unselected character attribute to each field. 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.

User Validation Routine

The user validation routine applies only if TypeForm controls form fill-in processing.

This option allows the application program to verify the value entered in a field, whenever this value is modified, rather than performing user-specific validation for the whole form after the TypeForm operation is complete. If any value is erroneous, it calls TypeForm again to request a modification from the user.

When defining the field, the user may associate with it the name of a user validation routine. All the user validation routines must be public procedures linked with the application program; you must declare them when configuring Forms Run-Time.

During form fill-in, each time the field contents are modified without the field becoming empty, TypeForm automatically calls the user validation routine (if it exists). The user validation routine receives as input the name, index, and number of the field, and its current value. If it validates the value, it can optionally return to TypeForm the name, index or number of the field that must be selected next for input. The default is to let TypeForm choose the next field to be selected, depending on the last function key the user has entered. If it does not validate the value, the field remains selected. The user validation routine specifies the cursor location in the field and the error message to be displayed.

Help Message

The help message displays on the bottom line of the screen each time the user presses the **Help** key while filling in a field.

Note: The Help key may be pressed while in the Forms Editor to bring up a single screen that lists the function key assignments and the character attributes table.

List of Values

A field can be created so that it can never contain anything but a value out of a limited list the user specified when defining the field.

When such a field is selected for input, the user can either:

- display one by one all the values in the list, using a Next value or Previous value function key
- enter any ASCII character, in which case the first value in the list starting with this character will display (the search starting with the value immediately following the one currently displayed, so that if two different values start with the same character, they can be selected one after the other)

The default value or any value written into the field by the application program is valid only if it is equal to one of the values in the list, or to the first characters of one of these values. In the second case, the whole value is displayed.

Uppercase and lowercase letters are considered as equivalent when matching a character entered by the user, the default value, or a value written by the application program with one of the values in the list.

When the DisplayForm operation initially displays the form, a field defined with a list of values contains its default value or, if no default value has been specified, the first value in the list. The show-default property is ignored. In any case, such a field can never be empty.

Selected/Unselected Color

If your workstation has color capability, you can choose the colors for lines, fields, and captions.

Color selection for a given form can be made from a color palette made up of eight colors, numbered 0 through 7. Each color is made up of component hues of red, green, and blue. Four intensities of each hue are possible, numbered 0 through 3. When the desired color and hue selections are made the palette becomes a permanent part of the form.

Displaying a Report on a Form (Forms Reporter)

The Forms Reporter (FREPORT command) displays information about a form (form name, size, height and width, and number of fields) and lists each field's defined characteristics. Refer to Display a Forms Report in section 3 for information.

- name
- position
- size
- single or repeating
- index range
- options

You can direct the form's characteristics and image to the screen, disk file, or print device.

Forms Run-Time

Through the Forms Editor, you can draw rulings of various styles, define text captions, and designate fields to be completed at run time. The completed form design is stored in a file, accessible at any time by the application program using Forms Run-Time.

Forms Run-Time, a library of object modules that can be linked to the application program, provides the following capabilities:

- displaying a previously-designed form
- obtaining information about a form
- allowing the user to enter data in the form
- returning data into the calling application program

In its library of object module procedures, Forms Run-Time has two major user-input routines: the UserFillField routine and the TypeForm routine. The UserFillField routine allows form fields to be completed one by one under program control. The TypeForm routine completes all form fields without requiring program intervention.

You can write a program in any of the programming languages available for workstations, and use Forms Run-Time services.

Using the UserFillField Routine

The sequence for using Forms Run-Time with the UserFillField routine is as follows:

1 The program specifies the form name and, optionally, a display location.

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 exit from each field as follows:

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

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

If the user enters 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.

Using the TypeForm Routine

The sequence for using Forms Run-Time with the TypeForm routine is as follows:

1 The program specifies the form name and, optionally, a display location.

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 Forms Designer.

4 Forms Run-Time prompts the user to enter data into each field.

5 The user presses a form-validation key, which allows Forms Run-Time to encode and validate field input values as it returns them to the calling program. The user can alternatively press the form-cancel key to return control to the program without accepting any modified values.

Note: TypeForm can optionally validate each field's entry each time the entry is changed (refer to section 3).

Opening a Form

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 report and displays at the top of the Forms Editor display (unless the form fills the display) so you can monitor form size as you create a form. A full-display form requires a work area of approximately 2Kb.

The program must open the form before a user can enter data in it. You use the Forms Run-Time OpenForm service to instruct the program to open the form (refer to section 4).

In addition, you can use the BTOS Linker 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 *BTOS II Language Development Linker and Librarian Programming Guide*.)

Note: Forms are easier to maintain if the form and run files are separate.

Custom Keyboard Configurations

You can reconfigure (or redefine) the keyboard functions by using the TabKey table contained in FmTabKey.asm or FmTabKey5.asm.

The Forms.Lib provided with the BTOS Forms 6.0 release has the Forms 4.0 keyboard table as the default configuration. If you wish to use the Forms 5.0 configuration, you must assemble the FmTabKey5.asm, renaming it to FmTabKey.obj during assembly. This module must be placed in the Forms.Lib using the BTOS Librarian. Your application(s) must be relinked with Forms.Lib to complete the procedure.

In this manual, a key that you can reconfigure is referred to by its function. For example, a key you use to move the cursor to the next field in a form is called the Next Field key.

Keys that are reserved and cannot be reconfigured are called by their key names (for example, **DELETE**, **BACKSPACE**, and **OVERTYPE**).

For more information on reconfiguring the keyboard, refer to section 4.

Using the Mouse

If the mouse server has been installed on your system, you can use the mouse to select areas of the screen. The mouse cursor is a solid block and moves independently from the normal cursor. When the left mouse button is pressed, the action is the same as for the **MARK** key. The function of the right mouse button is the same as for the **BOUND** key.

For additional information about mouse hardware and software use and installation, refer to the *BTOS II System Reference Manual*.

Software Installation

You use the procedures in this section to install your Forms software. After you install the software, you run the Forms Editor and Forms Reporter (FREPORT) by entering commands at the Executive level. In addition, the Forms Run Time Services are available for linking forms into run files.

You install the Forms software from the software diskettes. The diskettes are write-protected; you should not write-enable them or use them as a working copy.

The system directs the software installation, prompting you when it requires information.

The *BTOS II Standard Software Operations Guide* contains additional information on installation procedures.

The installation run file utilizes a set of forms to interactively install the Forms software.

Installing the Forms Software

To install the Forms software use the following procedure:

1 Sign on to your system.

2 Insert the first of the two software diskettes in floppy disk drive [f0].

3 Enter one of the following entries in the Executive command line:

   □ To install on a workstation, enter **SOFTWARE INSTALLATION.**

   □ To install on an XE520 master, enter **XESOFTWARE INSTALLATION.**

4 Press **GO.**

   A form is displayed.

5 Press **GO** to install the Forms 6.0 software (default is **I** (Install)).

Note: If you want to deinstall, press **D** to remove previously installed Forms files and commands. Then press **GO**.

The installation configuration form is then displayed. You can change the default values specified in this form to customize the installation of the Forms software for your workstation.

6 If your workstation is a B26, B28, or B38 with graphics hardware, and you want to use the graphics-capable Forms Editor, enter **YES** following the question **Do you want Editor Graphics Capability**

7 Press **GO** to continue the installation procedure. Press **CANCEL** to abort.

8 Based on the options you have chosen, the number of unused sectors for the specified volume and the total number of sectors needed for the chosen configuration are calculated and displayed.

9 If you have enough sectors, press **GO** to continue. If you do not have enough sectors, press **CANCEL** to halt. The installation continues when you press **GO**.

10 When the system prompts you, remove the first software diskette from your workstation's floppy disk drive and install the second diskette.

Note: If you selected No to install the Forms Library, installation is complete.

11 Press **GO**.

12 When the message *****INSTALLATION OF BTOS FORMS DESIGNER COMPLETE***** appears, remove diskette 2 of 2 and store it properly.

Table 2-1 lists the new commands you can enter from the Executive after you install the Forms software.

Table 2-1 **BTOS Forms Designer Executive Level Commands**

Command	**Description**
FORMS EDITOR	editor for modifying and creating forms
FREPORT	reporter for detailed information on form size, number of fields, and field characteristics

Forms Editor and Forms Reporter

The Forms Editor and Forms Reporter provide you with two tools for designing forms:

□ You can use the Forms Editor to design or modify a form. The Forms Editor includes a Test Drive function that allows Forms Run-Time to take control. Forms Run-Time displays the form and allows you to test the form as a user.

□ You can use the Forms Reporter to display statistics about the form, such as size and field characteristics.

This section contains procedures for using the Forms Editor and Forms Reporter. In addition, appendix D contains a training exercise for creating a form and a sample report.

Activating the Forms Editor

To activate the Forms Editor when you want to create a new form, enter **FORMS EDITOR** in the Executive command line; then press **GO.** The Forms Editor display appears.

Note: You do not have to specify a name for your form before you design it; however, if you want to save the form you must specify a name before exiting Forms Editor.

To activate the Forms Editor when you want to revise an existing form, use the following procedure:

1 Enter **FORMS EDITOR** in the Executive command line.

2 Press **RETURN**.

The system displays one of two **FORMS EDITOR** command forms, depending on whether graphics capability is installed (refer to figure 3-1 and figure 3-2).

3 Enter the file name in the **[File]** field.

Do not include a .form suffix; the system adds the suffix when you use the Write Form function (refer to Writing a Form).

The system uses the current volume and directory as the default path. To access a form from a different volume or directory, you must include the volume and directory names.

4 Enter the name of the form you want to modify in the **[Form]** field, if the file includes more than one form.

The default is the file name.

5 Enter the name of the file in the **[Graphic File]** field, if a graphics file is to be displayed in the background on graphics workstations only.

6 To display the form starting at a particular column from the upper left-hand corner, enter the starting-column coordinate in the **[Column]** field.

The default centers the form, left to right, on the display.

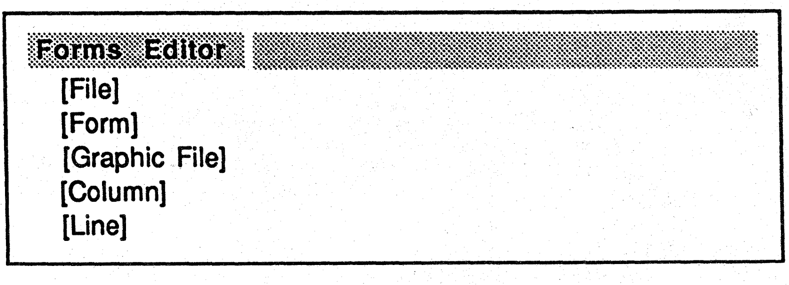
7 To display the form starting at a particular line from the top of the form, enter the starting-line coordinate in the **[Line]** field.

The default centers the form, top to bottom, on the display.

8 Press **GO**.

The Forms Editor displays the form.

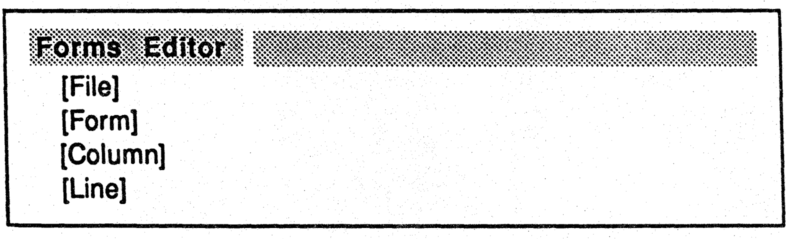
Figure 3-1 **Forms Editor Command Form with Graphics**



Forms Editor

[File]
[Form]
[Graphic File]
[Column]
[Line]

Figure 3-2 **Forms Editor Command Form without Graphics**



Forms Editor

[File]
[Form]
[Column]
[Line]

Forms Editor Display

The Forms Editor display includes two areas:

- □ a status area above the line
- □ a work area below the line

The system displays messages in the center of the status area. After you have named and saved the form, (refer to Writing a Form), 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 display area, the work area expands and the status information no longer displays.

When you activate the Forms Editor to create a form, the work area is blank except for the cursor; when you activate the Forms Editor to modify a form, the form appears in the work area with the cursor at the last line edited.

Moving the Cursor

The cursor may be moved by:

- □ pressing **Left Arrow**, **Right Arrow**, **Down Arrow**, or **Up Arrow** individually or in combination (refer to table 3-1).
- □ using the mouse, if you have the mouse server installed, by using the left mouse button.

Selections

A selection is a rectangular display area that you select (specify) by using **MARK** and **BOUND** or by using the mouse. After you select an area, you can use the Forms Editor functions on the selection (refer to Functions).

You can select only one area at a time; however, the selection can be any size and can include fields or a portion of a field, line, or caption.

Table 3-1 **Cursor Movement**

To Move The Cursor	**Press These Keys**
three lines up or down	**SHIFT** and **Up Arrow** or **Down Arrow**
five spaces right or left	**SHIFT** and **Right Arrow** or **Left Arrow**
to the display edge	**CODE** and **Up Arrow, Down Arrow, Right Arrow,** or **Left Arrow**
to the center	**Right Arrow** and **Left Arrow**; or **Up Arrow** and **Down Arrow**
diagonally	**Up Arrow** or **Down Arrow** and **Left Arrow** or **Right Arrow**
to a display corner	**CODE** and **Up Arrow** or **Down Arrow** and **Left Arrow** or **Right Arrow**

To make a selection, use the following procedure:

1 Position the cursor at one corner of the area you want to select.

2 Press **MARK.**

The character cell containing the cursor appears in reverse video. This becomes the marked corner of your selection; to change it, you must press **MARK** to remove the selection and then return to step 1 to select the marked corner.

3 Move the cursor to the diagonally opposite corner of the area you want to select.

This location becomes the bound corner of your selection.

4 Press **BOUND.**

The selected area appears as a reverse video rectangle.

To change the width or height of your selection, move the cursor to a different location and press **BOUND.**

The marked corner of the selection remains fixed, but the bound corner can be changed as often as you like to fine-tune your selection.

Press **MARK** to remove the selection. The character cell at the cursor location remains in reverse video.

Captions

The Forms software protects captions; lines cannot overwrite these text entries during forms editing, nor can the user modify them.

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.

Editing a caption does not affect the position of other captions on the same line. In the following example, changing left caption to left-hand caption does not change the position of right caption:

```
left caption                right caption
left-hand caption           right caption
```

If you edit one of these same line captions so that a caption runs into a caption to the right, or if you try to insert more characters than can fit on that line, the system beeps.

A caption overrides a selection. If you start entering a caption after making a selection, the selection disappears. You can recover the selection after you finish entering the caption by using the **Reselect** function.

Edit Modes

You use either insert or overtype mode to create or modify captions. In overtype mode, the characters you enter replace the characters at the cursor position. In insert mode, the characters you enter are inserted at the current cursor position (the character at the cursor position and the cursor move to the right).

For overtype mode, use the following procedure:

1 To activate the overtype mode, press **OVERTYPE**.
  The keycap light (LED) comes on.

2 To return to the insert mode, press **OVERTYPE** again.
  The LED goes off.

Note: The Forms Editor does not wrap text, but you may repeat fields to obtain similar results. If an insert causes the text to pass the edge of the display or to run into text that ends at the display edge, the system beeps and does not accept the entry. For repeating fields, text can be entered into a field until it is filled. Then the cursor must be moved to the next field and text can again be entered.

DELETE and BACKSPACE Keys

DELETE and **BACKSPACE** can help you edit captions. When you press **DELETE**, you remove one character (at the cursor position).

BACKSPACE functions differently depending on which edit mode you use, as follows:

□ In overtype, **BACKSPACE** is equivalent to **Left Arrow**; it moves the cursor to the left but does not delete text.

□ In insert, **BACKSPACE** moves the cursor to the left and deletes text to the left of the cursor position as it moves.

Literally Inserting Characters and Symbols

Press the CODE and the Equals (=) keys together to literally insert the next key you press. For example, to insert an Up Arrow character in your form, press **CODE-=**, then press **Up Arrow**.

All characters in the range 00h-FFh are accessible.

Forms Editor Functions

The Forms Editor includes functions that you use to create, revise, save, test, and use forms. You activate the Forms Editor functions by pressing function keys **F1** through **F10** and **HELP** (refer to table 3-2), using:

- □ **RETURN**, **NEXT**, **TAB**, or **Down Arrow** to move to the next field in a form
- □ **Up Arrow** to move to the previous field in a form
- □ **CODE-Up Arrow** to move to the first field in a form
- □ **CODE-Down Arrow** to move to the last field in a form
- □ **SCROLL UP** to move to the next value in a field
- □ **SCROLL DOWN** to move to the previous value in a field
- □ **GO** to validate a form
- □ **CANCEL** to cancel a form

Table 3-2 **Forms Editor Functions**

Key	Function	Use This Function To:
F1	Reselect	select the last selected area
F2	Undo	delete the last change
F3	Graphics*	display graphics
SHIFT-F3	Graphics*	undisplay (remove) graphics
F4	Draw	draw a line or box (single thin bar)
CODE-F4	Draw	draw a line or box (double thin bar)
SHIFT-F4	Draw	draw a line or box (single thick bar)
F5	Erase	remove a line or box
F6	Read Form	display a saved form
CODE-F6	Insert Text	display a saved text file
F7	Write Form	save a form
F8	Define Field	define field characteristics
CODE-F8	Color Selection	select the color palette for a given form
F9	Test Drive	test the form as a user
F10	Delete Selection	delete captions, lines, and fields from the selection
CODE-=	Insert	insert literal character
CODE-(A-P)	Define Attribute	define attribute of selected area (refer to table 3-3)
HELP	Forms Editor Help screen	display the Forms Editor Help Screen (refer to figure 3-3)

* Forms with graphics capability

Figure 3-3 shows the Forms Editor Help Screen as it appears when you press **HELP**. The Help Screen shows the function key assignments and the character attributes table.

Reselecting a Previously Selected Area (F1)

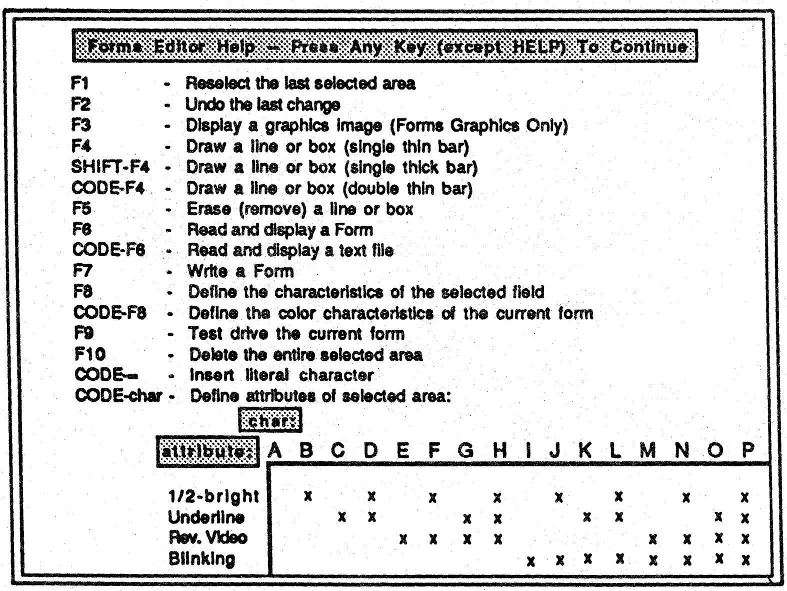
In most cases, when you select an area and then perform a function on that selection (for example, Draw), the Forms Editor removes the selection from that area. If you want to select the same area that the function was performed on, press **Reselect** (F1) and position the cursor to the receiving location.

Undoing the Most Recent Function (F2)

You can remove the effect of your most recent function by pressing **Undo** (F2). The Forms Editor also returns the selection and cursor to the previous position.

Undo restores deleted text or erased lines, deletes inserted or copied lines, and replaces moved material in its original position. Any uninterrupted sequence of text-entry operations counts as a single function for Undo.

Figure 3-3 Forms Editor Help Screen



Forms Editor Help -- Press Any Key (except HELP) To Continue

F1	- Reselect the last selected area
F2	- Undo the last change
F3	- Display a graphics image (Forms Graphics Only)
F4	- Draw a line or box (single thin bar)
SHIFT-F4	- Draw a line or box (single thick bar)
CODE-F4	- Draw a line or box (double thin bar)
F5	- Erase (remove) a line or box
F6	- Read and display a Form
CODE-F6	- Read and display a text file
F7	- Write a Form
F8	- Define the characteristics of the selected field
CODE-F8	- Define the color characteristics of the current form
F9	- Test drive the current form
F10	- Delete the entire selected area
CODE-–	- Insert literal character
CODE-char	- Define attributes of selected area:

char:

attribute:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1/2-bright		x		x		x		x		x		x		x		x
Underline			x	x			x	x			x	x			x	x
Rev. Video					x	x	x	x					x	x	x	x
Blinking									x	x	x	x	x	x	x	x

Displaying Graphics (F3)

If you have installed Forms Designer with graphics display capability, you can display a graphics picture that is contained in a file.

To display a graphics picture, use the following procedure:

1 Press **Graphics Display** (F3).

The system displays the Graphics Display form (refer to figure 3-4).

2 In the **[Graphics File]** field, enter the name (including the suffix) of the file containing the graphic picture (the default suffix is .pic).

If you already used the Graphics Display function, the name of the file you previously specified appears in the **[Graphics File]** field.

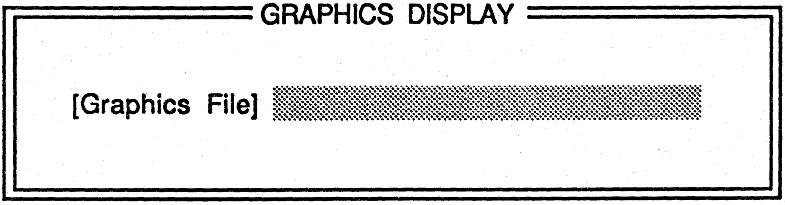
3 Press **GO**.

The system displays the graphics picture. If the picture is larger than the form, the picture overlaps the form's boundaries.

To remove a graphics picture, do the following:

1 Press **SHIFT-F3** to remove the displayed graphics picture.

Figure 3-4 **Graphics Display Form**



GRAPHICS DISPLAY

[Graphics File] ▨

Drawing Lines and Boxes (F4)

You can use the **Draw** function (**F4**) to draw lines and 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 display and then press:

- □ **Draw** (**F4**) for normal lines
- □ **SHIFT** and **Draw** (**F4**) for heavy lines
- □ **CODE** and **Draw** (**F4**) for double lines

To draw a vertical line, use the following procedure:

1 Select a rectangle with a width equal to one character cell.

2 Press **Draw** (**F4**) with or without **SHIFT** or **CODE**.

To draw a horizontal line, use the following procedure:

1 Select a rectangle with a height equal to one character cell.

2 Press **Draw** (**F4**) with or without **SHIFT** or **CODE**.

To draw a box, use the following procedure:

1 Select a rectangle with width and height greater than one character cell.

2 Press **Draw** (**F4**) with or without **SHIFT** or **CODE**.

The Forms Editor draws the lines you specified through the middle 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 (normal, heavy, double) does not have to correspond to the box lines.

Erasing Lines and Boxes (F5)

You can use the **Erase** function (**F5**) to remove lines or boxes. The Forms Editor does not erase captions within the selected area.

To erase lines or boxes, use the following procedure:

1 Select the line, box, or portion of line or box to be erased.

2 Press **Erase** (**F5**).

The system deletes all the lines within the boundary of your selected area. If the erased line intersects other lines, the Erase function clears the intersections.

Reading a Form (F6)

You use the **Read Form** function (**F6**) to recall a previously saved form after you activate the Forms Editor.

Caution: When the form or text displays, the Forms Editor overwrites the display, erasing any data. If it overwrites a form that you have not saved by using the Write Form function (refer to Writing a Form), that form is lost.

To read a form, use the following procedure:

1 Press **Read Form (F6)**.

The system displays the Read Form form (refer to figure 3-5). If a form is currently displayed, the form name appears in the **File:** field.

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

Do not include a **.form** suffix; the system adds the suffix when you use the Write Form function (refer to Writing a Form).

3 Enter the form name in the **[Form]** field, if the file has more than one form.

The default is the file name.

4 Enter the coordinates for the **[Column]** and **[Line]** fields of the **[Display]** line, if you do not want the form to appear in the center of the display (default).

5 Press **GO**.

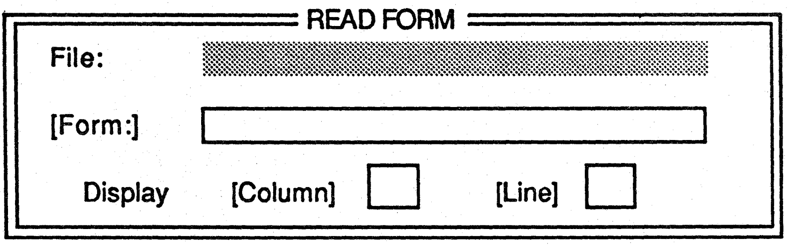
If an unsaved form is currently open, the Forms Editor prompts you to confirm that you want to erase it.

6 Choose one of the following:

- Press **GO** to erase the current form and display the form you requested.
- Press **CANCEL** for the Forms Editor to terminate the Read Form function.

Note: You can open a previously saved form when you first activate the Forms Editor (refer to Activating the Forms Editor).

Figure 3-5 **Read Form Form**



READ FORM

File: ▨

[Form:] □

Display [Column] □ [Line] □

You can use the **Read Form** function to recall the original version of an open form (if you have not yet saved the edited form under the same form name).

- □ To keep the revised version and display the original, use the **Write Form** function to save the revised form under a new form name. Then use Read Form to display the original version.
- □ To return to the original version (deleting your revisions), use the **Read Form** function while the revised form is displayed. When the system prompts you to confirm deleting the currently displayed form, press **GO**.

Inserting Text (CODE-F6)

You can use the **Text Insertion** function (**F6**) to recall a previously saved text file after you activate the Forms Editor. The text file appears in a field you select by using **MARK** and **BOUND** (described in Selections).

To insert text, use the following procedure:

1 Use **MARK** and **BOUND** to select an area.

2 Press **Text Insertion** (**CODE-F6**).

The system displays the Text Insertion form (refer to figure 3-6). If a form is currently displayed, the form's file name appears in the **[File]** field, suffixed by .txt.

3 Enter the text file name in the **[File]** field.

4 Enter the password in the **[Password]** field, if the text file is protected with a password.

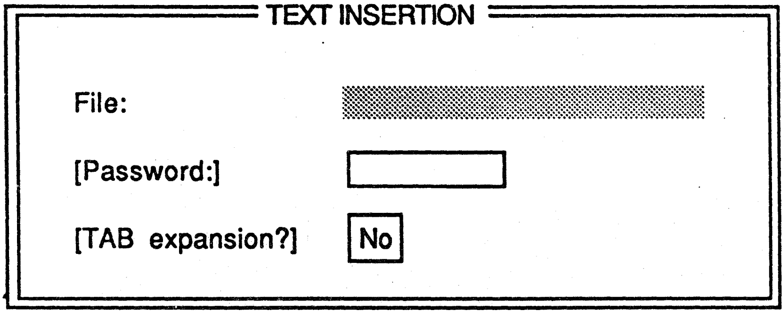
5 Enter the word **Yes** in the **[TAB expansion?]** field to replace tab characters with spaces.

If you do not want to replace tab characters with spaces, the ASCII representation for TAB characters displays.

6 Press **GO**.

The text file appears in the area selected with **MARK** and **BOUND**. All word processor attributes are ignored.

Figure 3-6 **Text Insertion Form**



TEXT INSERTION

File: ▨

[Password:] □

[TAB expansion?] No

Writing a Form (F7)

You use the **Write Form** function (**F7**) to save and name a displayed form. This function writes your form as an object module that the BTOS Librarian can store and the BTOS Linker can add to run files. The system gives the new object module a public label which is the same as the file name.

If you use the BTOS Librarian for your forms modules, any modification to your form must done by Forms Editor, and you must use BTOS Librarian to update the library.

Caution: You should save frequently. If you exceed the character attribute limitation, the Forms Editor exits and you lose all revisions you made since the last Write Form function.

To write a form, use the following procedure:

1 Press **Write Form (F7)**.

The Forms Editor displays the Write Form form (refer to figure 3-7).

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

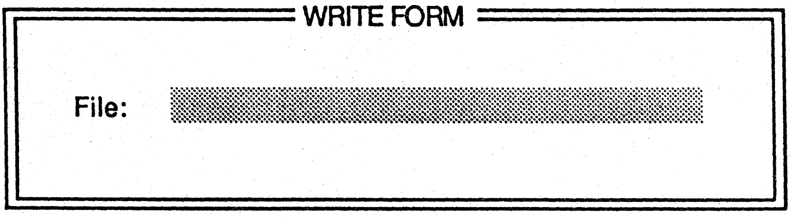
Do not include a **.form** suffix; the system adds this suffix automatically.

The system uses the current volume and directory as the default path. To specify a different volume or directory, you must include the volume and directory names.

3 Press **GO**.

The message **Writing...** appears. The Forms Editor adds a **.form** suffix to the file name (if the filename does not currently have a suffix), and translates the form into an object module. Status information appears in the status area unless the form is too large.

Figure 3-7 **Write Form Form**



WRITE FORM

File:

Defining Fields (F8)

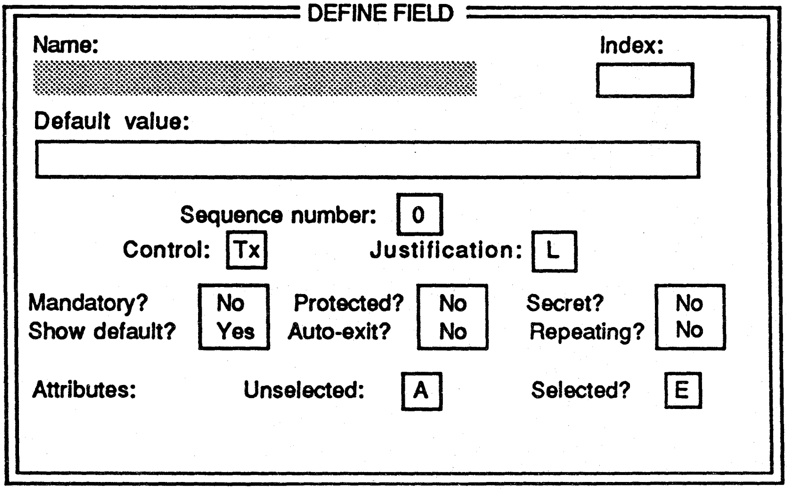
You use the **Define Field** function (**F8**) to add or revise fields and display the properties of individual fields.

Caution: You should save frequently. If you exceed the character attribute limitation, the Forms Editor exits and you lose all revisions you made since performing the last Write Form function.

Defining Single Fields

You can define a single field for any rectangular area that is exactly one character cell high. To define a single field, you use two forms (refer to figures 3-8 and 3-9).

Figure 3-8 **First Define Field Form**



DEFINE FIELD

Name:

Index:

Default value:

Sequence number: 0

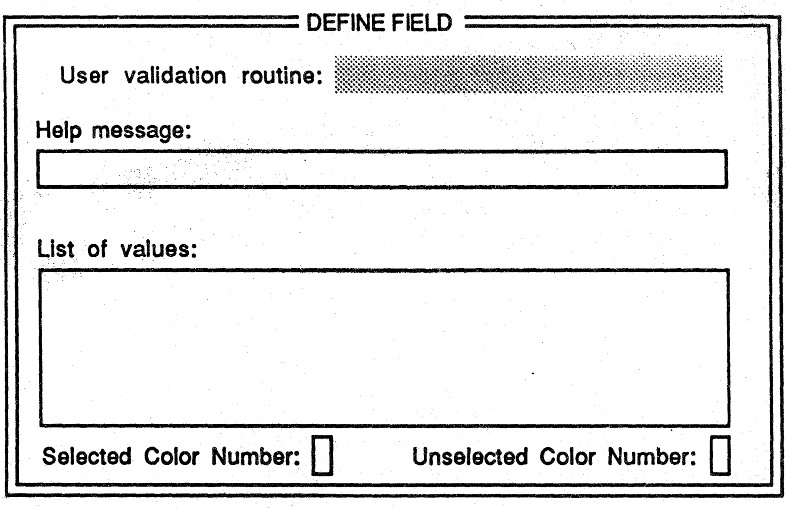
Control: Tx Justification: L

Mandatory? No Protected? No Secret? No

Show default? Yes Auto-exit? No Repeating? No

Attributes: Unselected: A Selected? E

Figure 3-9 **Second Define Field Form**



DEFINE FIELD

User validation routine:

Help message:

List of values:

Selected Color Number: □ Unselected Color Number: □

Completing the First Define Field Form for a Single Field

To complete the first Define Field form for a single field, use the following procedure:

1 Select the area using the cursor keys and **MARK** and **BOUND** (refer to Selections).

2 Press **Define Field (F8)**.

The system displays the first Define Field form (refer to figure 3-8). The cursor is in the **Name** field.

3 Enter a unique name for the field, up to a maximum of 40 characters, and press **RETURN**.

Note: Do not enter data in the **Index** field; the Forms Editor uses the **Index** field to identify repeating field locations (refer to Defining Repeating fields).

4 To assign a default value, enter the value (alphanumeric) in the **Default Value** field.

5 Press **RETURN**.

6 If you use TypeForm to complete form fields, move the cursor to the **Sequence number** field.

This field lets you tell TypeForm to fill fields in a particular sequence.

7 Specify the sequence number of the fields for TypeForm to complete.

The default is 0; this attribute tells TypeForm to fill fields starting at the top left side of the form and proceeding from left to right until the bottom of the form is reached. If you want TypeForm to fill fields in a different sequence, assign continuous numbers to each field, according to the sequence that TypeForm is to follow. For example, assign the number 1 to the first field in the sequence, assign the number 2 to the next field; continue numbering the entries sequentially, up to 200.

8 Move the cursor to the **Control** field.

9 Choose one of the following:

- □ Press **RETURN** to accept the default **TX**, which allows the field to accept any ASCII characters.
- □ If you want the field to accept characters other than ASCII characters, enter one of the following two-letter mnemonics:
  - □ **Ab** for uppercase and lowercase alphabetical letters only (that is, letters A through Z and a through z)
  - □ **Nu** for numbers 0 through 9 and for numeric symbols including the plus sign (+), minus sign (–), and decimal point (.) for decimal fields
  - □ **An** for alphanumeric characters including all 26 uppercase and lowercase alphabetical letters, and all numbers, spaces, and special characters (such as ! and #)

10 Move the cursor to the **Justification** field.

11 Choose one of the following:

- □ If you want left-justified field entries, accept the default (**L**) and make no entry.
- □ If you do not want left-justified field entries, enter one of the following mnemonics:
  - □ **R** for right-justified entries
  - □ a number between 1 and 15 that corresponds to the number of positions to the right of a decimal point

12 Move the cursor to the **Mandatory?** field.

13 Choose one of the following:

- □ Make no entry if you want this field to be optional (either filled in or left blank).
- □ Enter **Yes**, if you want the user to complete this field (with either text or numbers).

Note: You can specify a field as mandatory, protected, or optional; however, you cannot specify a field as both mandatory and protected.

14 Move the cursor to the **Protected?** field.

15 Choose one of the following:

- □ Make no entry if you want this field to be optional (accepting data from either a program or an operator).
- □ Enter **Yes**, if you want the field to accept data only from a program.

16 Move the cursor to the **Secret?** field.

17 Choose one of the following:

- □ Make no entry if you want the characters that a user or program inserts into a field to appear on the display.
- □ Enter **Yes**, if you do not want the characters that a user or program inserts into a field to appear on the display. The characters a user or program inserts into the field then appear as number signs (#) on the screen.

18 Enter **No** in the **Show Default?** field to prevent Forms Run-Time from displaying the default.

19 Enter **Yes** in the **Auto-exit?** field to set up an automatic exit from the field.

Note: If the **Repeating?** field is set to **Yes**, your selection was for more than one field. Press **CANCEL** to terminate the Define Field function, or follow the procedure for repeating fields (refer to Defining Repeating Fields).

20 Choose one of the following:

- □ Make no entry if you want the default attributes.

  The default attributes are:

  - □ **A** for unselected (Forms Run-Time adds no attributes when the cursor is not in the field)
  - □ **E** for selected (Forms Run-Time displays the field in reverse video when the cursor is in the field)
- □ Enter new entries, if you want to change the attributes (refer to table 3-3).

Table 3-3 **Character Attributes**

LETTER		**CHARACTER ATTRIBUTE**		
	Blinking	**Reverse Video**	**Underline**	**Half-bright**
A	—	—	—	—
B	—	—	—	yes
C	—	—	yes	—
D	—	—	yes	yes
E	—	yes	—	—
F	—	yes	—	yes
G	—	yes	yes	—
H	—	yes	yes	yes
I	yes	—	—	—
J	yes	—	—	yes
K	yes	—	yes	—
L	yes	—	yes	yes
M	yes	yes	—	—
N	yes	yes	—	yes
O	yes	yes	yes	—
P	yes	yes	yes	yes

Completing the Second Define Field Form for a Single Field

To complete the second Define Field form for a single field, use the following procedure:

1 Press **NEXT PAGE** to display the second Define Field form (refer to figure 3-9).

The cursor is in the **User validation routine** field.

Note: To redisplay the first Define Field form, press **PREV PAGE**.

2 If you use TypeForm and a validation routine, enter the name of the validation routine in the **User validation routine** field, up to a maximum of 30 characters.

Each time a user inserts new entries into a form's fields, TypeForm lets the program use a validation routine to make sure all form fields contain proper information; if the validation routine finds an improper field value, it displays an error message and the field that contains the improper value. (For more information about user validation routines, refer to section 4.)

3 Move the cursor to the **Help message** field.

4 Enter the Help message, up to a maximum of 60 characters.

The Help message appears at the bottom of the screen, if the user presses **HELP** when filling in a form field.

5 Move the cursor to the **List of values** field.

This field lets you assign up to eight values to a particular field; each value can contain up to 60 characters. The operator then selects a value for this field from the list of available values by either:

□ displaying each value one by one. The operator uses the function keys defined as Next Value and Previous Value keys to view the next value or the previous value, respectively, in the list. The Next Value function key is usually **SCROLL UP** and the Previous Value key is usually **SCROLL DOWN**; however, these functions can be reassigned to other keys (refer to section 4).

□ entering any ASCII character, which displays the listed value whose first character is that ASCII character. If two or more values in the list start with the same character, reentering the same ASCII character displays the next listed value that starts with the same character.

Note: The default value or any value that the program writes to this field must be equal either to one of the values in the list, or to the first ASCII character of one of the values in the list.

6 If your workstation has color capability, you can choose the colors for lines, fields, and captions.

Press **CODE-F8** to display the color palette form, which shows the color numbers 0 through 7.

Press **Define Field** (**F8**) to display the first form of the Define Field. Then press the **Next Page** key to display the second form of the Define Field, where the selected color number and unselected color number field entries appear as the last two entries on the form. The selected color number represents the color selected for the current field in which the cursor is located. The unselected color number represents the color selected for the field in which the cursor does not reside.

7 Press **GO**.

The form reappears. A square box (called a tag) appears for each character cell in the field (refer to figure 3-10). If you define two adjacent fields (with no separating captions or lines), one field contains filled boxes so you can distinguish between the two fields.

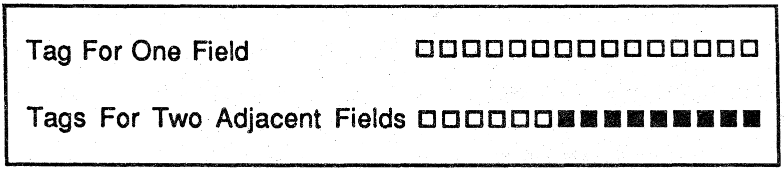
Defining Repeating Fields

It is simpler to define repeating fields that are stacked vertically; 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** (refer to Copying and Moving) to position each field, or you can define each repeating field separately.

To define a repeating field, you use both Define Field forms (refer to figures 3-8 and 3-9).

Figure 3-10 **Field Tags (Sample)**



Tag For One Field □□□□□□□□□□□□□□□

Tags For Two Adjacent Fields □□□□□□■■■■■■■■■

Completing the First Define Field Form for Repeating Fields

To complete the first Define Field form for repeating fields, use the following procedure:

1 Select the area using the cursor keys and **MARK** and **BOUND** (refer to Selections).

- □ For vertically stacked repeating fields, select all fields.
- □ For horizontal or random repeating fields, select the first field.

2 Press **Define Field (F8).**

The system displays the first of two Define Field forms (refer to figure 3-8). The cursor is in the **Field Name** field.

3 Enter a unique name for the field, up to a maximum of 40 characters.

You must use the same field name for each field.

4 Move the cursor to the **Index** field.

5 Choose one of the following:

- □ For vertically stacked fields, leave the default (**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.

Note: The remaining field characteristics are optional. For vertically stacked fields, you initially give each field the same characteristics. You can later select each field and modify these characteristics.

6 To assign a default value, enter the value (alphanumeric) in the **Default Value** field.

The default is an empty field (null). If the default you enter occupies more character cells than the field size, the Forms Editor shortens the default value (from the right) to fit.

7 If you use TypeForm to complete form fields, move the cursor to the **Sequence number** field.

This field lets you tell TypeForm to fill fields in a particular sequence.

8 Specify the sequence number of the fields for TypeForm to complete.

The default is 0; this attribute tells TypeForm to fill fields starting at the top left side of the form and proceeding from left to right until the bottom of the form. If you want TypeForm to fill fields in a different sequence, assign continuous numbers to each field, according to the sequence that TypeForm is to fill the fields. For example, assign the number 1 to the first field in the sequence, assign the number 2 to the next field; continue numbering the entries sequentially, up to 200.

9 Move the cursor to the **Control** field.

10 Choose one of the following:

- □ Press **RETURN** to accept the default **TX**, which allows the field to accept any ASCII characters.
- □ If you want the field to accept characters other than ASCII characters, enter one of the following two-letter mnemonics:
  - □ **Ab** for uppercase and lowercase alphabetical letters only (that is, letters A through Z and a through z)
  - □ **Nu** for numbers 0 through 9 and for numeric symbols including the plus sign (+), minus sign (–), and decimal point (.) for decimal fields
  - □ **An** for alphanumeric characters including all 26 uppercase and lowercase alphabetical letters, all numbers, spaces, and special characters (such as ! and #)

11 Move the cursor to the **Justification** field.

12 Choose one of the following:

- □ Make no entry to accept the default (**L**), which left-justifies field entries.
- □ If you do not want left-justified field entries, enter one of the following mnemonics:
  - □ **R** for right-justified entries
  - □ a number between 1 and 15 that corresponds to the number of positions to the right of a decimal point

13 Move the cursor to the **Mandatory?** field.

14 Choose one of the following:

- □ Make no entry if you want this field to be optional (either filled in or left blank).
- □ Enter **Yes**, if you want the user to complete this field (with either text or numbers).

Note: You can specify a field as mandatory, protected, or optional; however, you cannot specify a field as both mandatory and protected.

15 Move the cursor to the **Protected?** field.

16 Choose one of the following:

- □ Make no entry if you want this field to be optional (accepting data from either a program or an operator).
- □ Enter **Yes**, if you want the field to accept data only from a program.

17 Move the cursor to the **Secret?** field.

18 Choose one of the following:

- □ Make no entry if you want the characters that a user or program inserts into a field to appear on the display.
- □ Enter **Yes**, if you do not want the characters that a user or program inserts into a field to appear on the display. The characters a user or program inserts into the field then appear as number signs (#) on the screen.

19 Enter **No** in the **Show Default?** field to prevent Forms Run-Time from displaying the default.

20 Enter **Yes** in the **Auto-exit?** field to set up an automatic exit from the field.

21 In the **Repeating?** field:

- □ Leave the **Repeating?** option set to **Yes**, if you selected an area for vertically-stacked repeating fields.
- □ Enter the word **Yes**, if you selected one field of a repeating field.

22 Choose one of the following:

- □ Make no entry if you want the default attributes.

  The default attributes are:

  - □ **A** for unselected (Forms Run-Time adds no attributes when the cursor is not in the field)
  - □ **E** for selected (Forms Run-Time displays the field in reverse video when the cursor is in the field)
- □ Enter new entries, if you want to change the attributes (refer to table 3-3).

Completing the Second Define Field Form for Repeating Fields

To complete the second Define Field form for repeating fields, use the following procedure:

1 Press **NEXT PAGE** to display the second **Define Field** form (refer to figure 3-9).

The cursor is in the **User validation routine** field.

Note: To redisplay the first **Define Field** form, press **PREVPAGE**.

2 If you will be using TypeForm and a validation routine, enter the name of the validation routine in the **User validation routine** field, up to a maximum of 30 characters.

Each time a user inserts new entries into a form's fields, TypeForm lets the program use a validation routine to make sure all form fields contain proper information; if the validation routine finds an improper field value, it displays an error message and the field that contains the improper value. For more information about user validation routines, refer to section 4.

3 Move the cursor to the **Help message** field.

4 Enter the Help message, up to a maximum of 60 characters.

The Help message appears at the bottom of the screen, if the user presses **HELP** when filling in a form field.

5 Move the cursor to the **List of values** field.

This field lets you assign a list of up to eight values to a particular field; each value can contain up to 60 characters. The user then selects a value for this field from the list of available values by either:

- displaying each value one by one. The user utilizes the function keys defined as Next Value and Previous Value keys to view the next value or the previous value, respectively, in the list. The Next Value function key is usually **SCROLL UP** and the Previous Value key is usually **SCROLL DOWN**; however, these functions can be reassigned to other keys (refer to section 4).
- entering any ASCII character, which displays the listed value whose first character is that ASCII character. If two or more values in the list start with the same character, reentering the same ASCII character displays the next listed value that starts with the same character.

> **Note:** The default value or any value that the program writes to this field must be equal either to one of the values in the list, or to the first ASCII character of one of the values in the list.

6 Press **CODE-F8** to display the color palette form, which shows the color numbers 0 through 7.

Press **Define Field** (**F8**) to display the first form of the Define Field. Then press the **Next Page** key to display the second form of the Define Field, where the selected color number and unselected color number field entries appear as the last two entries on the form. The selected color number represents the color selected for the current field in which the cursor is located. The unselected color number represents the color selected for the field in which the cursor does not reside.

7 Press **GO**.

The form reappears. A square box (called a tag) appears for each character cell in the field (refer to figure 3-10). If you define two adjacent fields (with no separating captions or lines), one field contains filled boxes so you can distinguish between the two fields.

8 If you are creating horizontal or random fields, you must either:

- □ move each field to its location (be careful to keep the indexes sequential, left to right then top to bottom)
- □ repeat this procedure to define each field

Modifying Field Definitions

You use the **Define Field** function (**F8**) 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.

To modify field definitions, use one of the following:

- □ For Single Fields, select the field and use the Define Field function.
- □ For vertically stacked Repeating Fields, select an individual field or all the fields and use the Define Field function.
- □ For horizontal or random Repeating Fields, you must select each field and use the Define Field function.

Modifying Field Position and Width

You 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 **DELETE** to shorten the field (and move it to the left).
- Move or copy fields (refer to Copying and Moving).

If you split the tag, the first part of the tag becomes the field and the second part becomes a caption whose appearance is a sequence of square boxes.

Color Selection (CODE-F8)

You can select the color palette to be used with a given Forms Editor form with **CODE-F8**. The color palette includes eight colors, numbered 0 through 7. The palette in current use is displayed and changes to it are dynamically updated. Each color is made up of component hues of red, green, and blue. Each hue has four intensities, numbered 0 through 3. The color numbers to be used for Captions (text) and for Lines (and Boxes) may also be selected on this display.

You use the **NEXT** or **RETURN** key to move the active cursor (indicated by a flashing reverse video block) from area to area. Downward movement only is permitted, with wraparound from the bottom to the top. Within each area, cursor movement up or down is achieved with the **Up Arrow** and **Down Arrow** keys.

After you have selected the desired color palette, press **GO**. To abort this selection, press **CANCEL**.

Testing Forms (F9)

The **Test Drive** function (**F9**) simulates a form's field responses during run time and is similar to having a program call the TypeForm routine.

To test your form, use the following procedure:

1 Press **Test Drive** (**F9**).

The Forms Editor translates the form into the binary format of a form file (however, no file is written).

Forms Run-Time displays the form as it appears at run time and positions the cursor in the first field.

2 Try out your form by entering text, pressing the Next Field key to move the cursor to the next field, and pressing the Previous Field key to move to the previous field.

As you exit each field, Forms Run-Time reads and displays your entry. You can verify the character attributes, defaults, and auto-exit. If you press a key unknown to Forms Run-Time, a message displays.

Note: The Test Drive function does not support a First Field and Last Field key, a Form Validation and Form Cancel key, a Correction key, and a user validation routine. Furthermore, the Test Drive function performs data type control (that is, it verifies alphabetic, numeric, alphanumeric, and text entries) according to the standard keyboard configuration (refer to section 4); however, it does not perform data type control on default values.

3 Press **Test Drive** (**F9**) again to return control to the Forms Editor.

Field tags reappear.

Deleting Selections (F10)

You use the **Delete Selection** function (**F10**) to delete (erase) all captions, lines, and fields from the area you have selected.

To delete selections, use the following procedure:

1 Select the area to be deleted using the cursor and **MARK** and **BOUND**.

2 Press **Delete Selection** (**F10**).

The Forms Editor replaces the selected area with empty spaces.

Note: You can also press **CODE-DELETE** to access the Delete Selection function.

You use **Erase** (**F5**) to delete lines only (not text).

Copying and Moving

You copy or move lines, fields, or captions from one part of the form to another by selecting the area to be copied and using either **COPY** or **MOVE**. You also move or copy a text file recalled with the **CODE-F6** keys (refer to Inserting Text).

To copy or move lines, fields, or captions, use the following procedure:

1 Select the area to be copied or moved (the source area).

2 Position the cursor at the top left corner of the area to receive the lines, field, or 'captions (the target area).

  Source and target areas can overlap.

3 Choose one of the following:

  - □ To copy the source area to a target area using the cursor position as the top left corner of the target area, press **COPY.**
  - □ To move the source area to a target area using the cursor position as the top left corner of the target area, press **MOVE.**
  - □ To copy the source area to a target area using the cursor position as the top right corner of the target area, use **SHIFT-COPY.**
  - □ To move the source area to a target area using the cursor position as the top right corner of the target area, use **SHIFT-MOVE.**
  - □ To copy the source area to a target area using the cursor position as the top center of the target area, use **CODE-COPY.**
  - □ To move the source area to a target area using the cursor position as the top center of the target area, use **CODE-MOVE.**

  The system shortens lines extending across the edges of the copy or joins them to the surrounding lines.

  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. You can repeat the MOVE or COPY function.

Note: 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 to avoid duplicate fields.

If you move repeating fields, the indexes must remain sequential. You should edit the index numbers if necessary.

If you Test Drive a form with duplicate single field names or nonsequential repeating fields, the Forms Editor assigns the blinking character attribute to all fields in conflict.

Displaying Text in 132-Column Format (CODE-Z)

On B27 workstations, you can use the **Zoom** function to display text in 132-column format. Zoom changes the display from 80 to 132 columns, and back again. To switch modes from 80 to 132 columns or from 132 to 80 columns, you press **CODE-Z.**

If you attempt to zoom from 132 to 80 columns when the form in the work area is more than 80 columns wide, Zoom displays the message **Form is too wide** and does not change the display.

Displaying Text in 146-Column Format (CODE-Z)

On B26, B28, and B38 workstations with bit-mapped graphics, text can be displayed in 146-column format using the **Zoom** function. Zoom changes the display from 80 to 146 columns, and back again. To switch modes from 80 to 146 columns or from 146 to 80 columns, you press **CODE-Z.**

If you attempt to zoom from 146 to 80 columns when the form in the work area is more than 80 columns wide, Zoom displays the message **Form is too wide** and does not change the display.

Note: Because bit-mapped graphics does not support blinking attributes, blinking is indicated by outline characters or as selected by the user via the configuration file.

Viewing Edit Codes (CODE-V)

The **View Edit Codes** function makes all edit (formatting) codes visible on the display. Press **CODE-V** to view formatting. Press **CODE-V** again to exit.

Note: The Forms Editor deletes space characters you enter with the literal insert procedure when you exit the **View Edit Codes** function.

The Forms Editor cannot distinguish between the space formatting codes that the system makes visible with the **View Edit Codes** function and literal insert space characters.

Selecting Tags (NEXT)

You press **NEXT** to select the next field tag; this is an easy way to move through a form you are editing.

When you press **NEXT**, the Forms Editor starts at the cursor position and scans the display row by row, searching for a tag. If the cursor is currently in a tag, the search begins just beyond that tag. If the system does not find a tag between the cursor and the bottom of the display, the search resumes at the top of the display.

When the Forms Editor finds a tag, the Forms Editor positions the cursor in the first character cell of that tag.

Exiting the Forms Editor

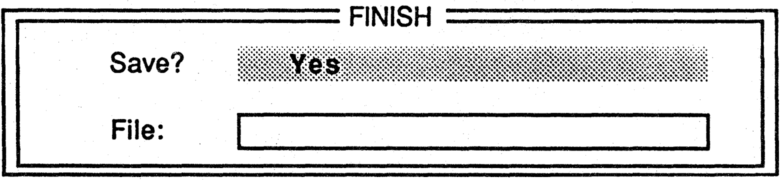
You press **FINISH** to end a Forms Editor session and return to the Executive.

If you have not used the **Write Form** function to save the form you edited or created during the session (refer to Writing a Form), the system displays the Finish form (refer to figure 3-11).

If you are revising an existing form, the **File** field default is the original file name. Your options are:

- To save the revised form under the original file name, press **GO**.
- To save the revised form in a different file, edit the **File** field and press **GO**.
- To delete the form, change the **Save?** field to **No**; then press **GO**.
- To interrupt the Finish operation and return to the Forms Editor, press **CANCEL**.

Figure 3-11 **Finish Form (Sample)**



FINISH

Save? Yes

File:

Displaying a Forms Report

The Forms Reporter (FREPORT command) provides the following information about a form you specify:

- name
- size in bytes, and height and width when displayed
- number of defined fields
- each field's:
  - name
  - row and column number (the top left corner has the coordinates 0,0)
  - width in characters
  - repeating characteristic
  - index
  - control characteristic
  - justification characteristic
  - sequence number
  - secret characteristic
  - protected and mandatory characteristics
  - default value
  - show default characteristic
  - auto-exit characteristic
  - selected and unselected character attributes
  - validation routine
  - help message
  - list of values
  - selected/unselected color number

Rather than displaying the report, you can instruct the system to write it to a disk file or printer.

To display a forms report, use the following procedure:

1 Enter **FREPORT** in the Executive command line.

2 Press **GO**.

The FREPORT command form appears (refer to figure 3-12).

3 Enter the file name in the **File** field.

If the system added the default suffix **.form** when you created the form, do not include it; otherwise, enter the entire file name.

4 If the file is a library file containing more than one form, enter the form name in the **[Form]** field.

5 Enter **No** in the **[Fields?]** field to prevent the system from reporting on the fields in the form.

The default is **Yes**.

6 Enter **No** in the **[Form image?]** field, if you do not want the form image to appear in the forms report.

The default is **Yes**.

7 Enter a file name or device name in the **[Output]** field to file or print the report.

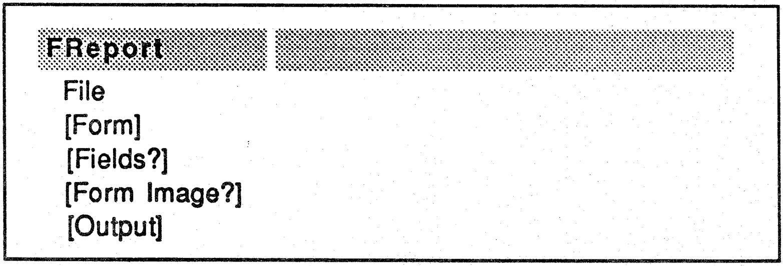
The default is display. If your form has too many fields for the entire report to fit on your display, you may want to print the report or specify a file.

8 Press **GO**.

A forms report displays. If the entire report does not fit on your display, press **NEXT PAGE** or **SCROLL UP** to display the rest of the form. When the system displays the last item of the report, the Forms Reporter exits to the Executive.

Refer to appendix D for a sample report.

Figure 3-12 **FREPORT Command Form**



FReport

File

[Form]

[Fields?]

[Form Image?]

[Output]

Creating Forms Library Files

You use the BTOS Librarian utility (refer to the *BTOS II Language Development Linker and Librarian Programming Guide*) to bundle forms in a library file of object modules.

When you create the forms for inclusion in a library file, follow these guidelines:

- make the file name and the form name the same
- avoid including an extension on the file name and the form name (the system default adds the suffix **.form**)
- do not add more than one extension to a file name or form name

After you create the forms, you use the BTOS Librarian utility to create a library. The library file name should have an extension to allow the Forms Editor and the Forms Reporter to access the forms.

To create a forms library file, use the following procedure:

1 Activate the BTOS Librarian by entering **LIBRARIAN** in the Executive command line.

The system displays the LIBRARIAN command form.

2 Enter the library name in the **Library file** field.

You create a new library by entering a new library name, or add new modules to a library by entering the name of an existing library.

3 Enter the form names in the **[Files to add]** field. Add the form names with all extensions. The BTOS Librarian automatically removes any suffixes after the first dot (.), and adds the suffix **.obj**.

4 Complete the LIBRARIAN command form.

5 Press **GO**

To revise a form contained in a forms library file, or to add or delete a form, you use the BTOS Librarian utility.

You display the forms contained in the forms library using the FORMS EDITOR command or the FREPORT command. You enter the library file name in the **File** field and the form name (without the suffix) in the **Form** field of the command form.

Caution: You cannot use the Forms Editor to revise forms contained in a forms library. After revisions are made, the revised form must be placed in the library with the BTOS Librarian to overwrite the original form.

Forms Editor Error Response

Users occasionally make errors that can be detected only when using **FINISH** to complete a form, using **F7** to write a form, or using **F9** to test drive a form. Common errors are repeating a field's name on a form, sequencing some (not all) fields on a form, sequencing fields with nonsequential (noncontinuous) numbers, and creating forms that take up too much memory.

Forms Editor indicates fields that contain such errors by making the fields blink; the user can then utilize **F2** to undo the blinking fields (refer to Undoing the Most Recent Function).

Forms Editor In-Memory Work Area

The Forms Editor uses an in-memory work area to store and access all information pertaining to a form you are editing. The size of this form must not exceed the work area (8K). The size of a form is based on:

- number of defined fields
- length of the name and default value of each field
- number and length of each caption
- number and length of the lines

You determine the exact number of bytes needed to represent a form by using the Forms Reporter. Approximate byte sizes are:

- each form requires a fixed overhead of 15 bytes
- each field requires 32 bytes

- each unique field name requires $(f+2)$ bytes (f is the length of the field name in bytes)
- each unique pair (field name, default value) requires $(d+3)$ bytes (d is the length of the default value in bytes). Fields with null default values are included.
- each caption requires $(c+4)$ bytes (c is the length of the run).

A run is a sequence of horizontally continuous character cells that contain the same character and video attributes. The two most common instances of a run are line characters and field definitions. A run is also defined for each horizontally contiguous sequence of null characters (characters that are not part of any line, caption, or field definition).

The Forms Editor uses another in-memory work area to store currently defined field names and default values. This work area contains 900 bytes. Each field requires $(f+d+1)$ bytes (f is the length of the field name; d is the length of the default value).

Forms Run-Time

Forms Run-Time consists of a library of object module procedures (forms.lib) you use to run forms with programs written in any of the programming languages available for workstations. Forms Run-Time also supplies four source modules for custom configuration for forms.lib.

Note: This section will help the programmer who is already familiar with the workstation operating system. Refer to the *BTOS II System Reference Manual* and the *BTOS II Customizer Programming Guide* for information about the workstation operating system.

Section 4 discusses Forms Run-Time services in alphabetical order and includes the following for each:

- an interface parameter table
- where applicable, a table of information returned by the service
- where applicable, a figure illustrating the buffer structure used by the service

AddValues

AddValues adds or modifies default values which you created when defining a field through the **Define Field** (**F8**) function key. It reads the initial descriptor from either a library or a single Form file, and writes the result to a new Form file.

The procedural interface is:

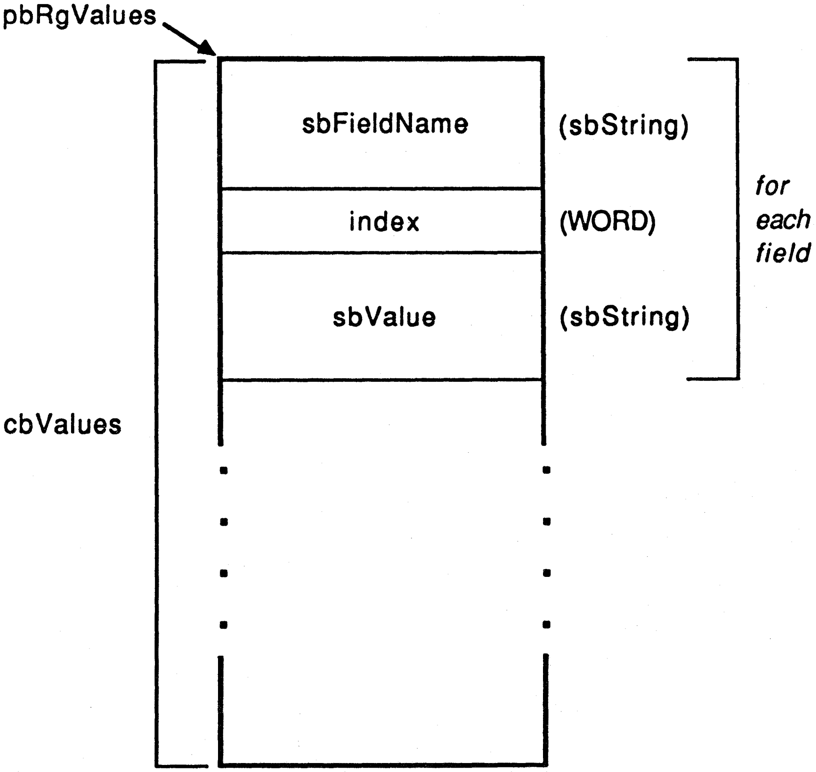
AddValues (pbFileName, cbFileName, pbFormName, cbFormName, pbRgValues, cbValues, pbNewFileName, cbNewFileName): ErcType.

Refer to table 4-1 for names and descriptions of AddValues procedural interface parameters.

Table 4-1 **AddValues Procedural Interface Parameters**

Name	Description
pbFileName cbFileName	pbFileName and cbFileName describe a character string containing the name of the Form file (or Library file).
pbFormName cbFormName	pbFormName and cbFormName describe a character string containing the name of the form to be read. If cbFormName is 0, the form name is the file name minus the file name extension, if there is one.
pbRgValues cbValues	pgRgValues and cbValues describe a user-supplied buffer containing information on the fields to be modified. Its structure must take the form shown in figure 4-1. You can specify field names in any order. The buffer can contain more or fewer field names than defined in the Forms descriptor.
pbNewFileName cbNewFileName	pbNewFileName and cbNewFileName describe a character string containing the name of the Form file to be created after modification. If the file already exists, AddValues appends the suffix -New.

Figure 4-1 **AddValues Buffer Structure**



DefaultField

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

The procedural interface is:

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

Refer to table 4-2 for names and descriptions of DefaultField procedural interface parameters.

Table 4-2 **DefaultField Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
pbFieldName cbFieldName	pbFieldName and cbFieldName describe a character string naming the field to be defaulted. In matching field names, DefaultField ignores the distinction between uppercase and lowercase.
index	Index specifies the field of a repeating field to be defaulted. DefaultField ignores this parameter, if the named field is not repeating.

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. You can center the form 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

Refer to table 4-3 for names and descriptions of DisplayForm procedural interface parameters.

Table 4-3 **DisplayForm Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
iFrame	iFrame is a frame number in the Video Control Block (refer to the *BTOS II System Reference Manual*).
iCol	iCol is a column number within the frame.
iLine	iLine is a line number within the frame.

EditForm

EditForm either displays a form on the screen or writes it to a file or device. It opens the form file and loads the form into memory, if required.

Optionally, EditForm can initialize the fields in the form with values the application program supplies. The type code it uses in decoding the data is always **Character**.

If the EditForm operation is called for writing a form to a file or device, it replaces all rulings with asterisks and other special characters in text captions or fields with spaces. It opens the output file as a byte stream in write mode, which means that it overwrites a previously existing file of the same name.

The procedural interface is:

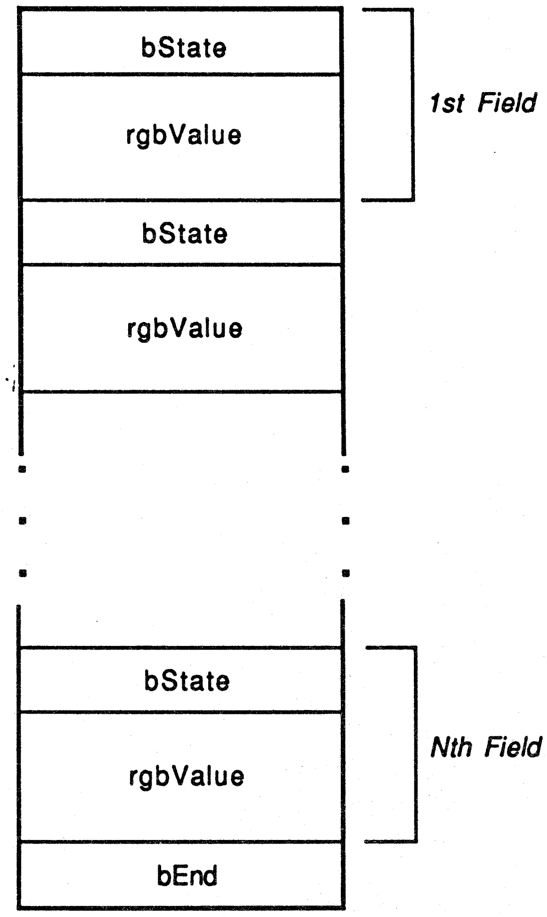
EditForm (pForm, cbMaxForm, pbFileName, cbFileName, pbFileDest, cbFileDest, pbFormName, cbFormName, iFrame, iCol, iLine, pbInit, cbInit): ErcType

Refer to table 4-4 for names and descriptions of EditForm procedural interface parameters.

Table 4-4 **EditForm Procedural Interface Parameters**

Name	Description
pForm cbMaxForm	pForm and cbMaxForm describe the work area of an already open form, or the work area into which EditForm must load the form.
pbFileName cbFileName	pbFileName and cbFileName describe a character string containing the name of the form file (or library file). If cbFileName is 0, no disk file is accessed: either the application program or a previous call to the EditForm operation is supposed to have already loaded the form into the work area.
pbFileDest cbFileDest	pbFileDest and cbFileDest describe a character string containing the name of the output file or device. If cbFileDest is 0, EditForm displays the form on the screen using the DisplayForm operation. If cbFileDest is not 0, it writes the form to the specified file or device using the Sequential Access Method.
pbFormName cbFormName	pbFormName and cbFormName describe a character string containing the form name of the form to be read. EditForm ignores these parameters, if it is not required to load the form from disk into the work area. If cbFormName is 0, the form name is the file name minus the file name extension, if there is one.
iFrame	iFrame is the number of the frame in which EditForm is to display the form. It ignores iFrame, if it is to write the form to a file or device.
iCol iLine	iCol and iLine determine the location within the frame at which EditForm is to display the form. You can center the form horizontally and/or vertically within the frame by specifying 255 for iCol and/or iLine. EditForm ignores these parameters, if it is to write the form to a file or device.
pbInit cbInit	pbInit and cbInit describe a buffer that contains the initialization values for the different fields of the form. If cbInit is 0, EditForm writes default values into the fields. The buffer's structure must take the form shown in figure 4-2.

Figure 4-2 **EditForm Buffer Structure**



In figure 4-2:

bState is a byte that can have any of the following values: 00 or 02 = the next bytes containing the initialization value for this field; 01 or 03 = no initialization value for this field.

There must be a bState byte for each field of the form. If bState is 01 or 03, the bState byte for the next field must immediately follow it.

rgbValue is a string of characters EditForm is to use as the initialization value for the field. If the field is decimal, the initialization value must include the decimal mark, except if the decimal part is empty. rgbValue must immediately follow a bState byte containing 00 or 02. Every character in the initialization value must be greater than 03.

bEnd is a byte that marks the end of the buffer. The value can be 00, 01, 02 or 03.

If a user has defined a specific sequence for this form, the elements (bState[,rgbValue]) for the different fields of the form must appear in the buffer in the order of the sequence numbers; if you have not defined a sequence, the elements must appear according to the location of the fields on the screen (from top to bottom and from left to right).

ExtractValues

ExtractValues returns a buffer containing the default values for each field of a form.

The procedural interface is:

ExtractValues (pbFileName, cbFileName, pbFormName, cbFormName, pbRgValuesRet, cbMax, pcbValuesRet): ErcType

Refer to table 4-5 for names and descriptions of ExtractValues procedural interface parameters.

Table 4-5 **ExtractValues Procedural Interface Parameters**

Name	Description
pbFileName cbFileName	pbFileName and cbFileName describe a character string containing the name of the Form file (or Library file).
pbFormName cbFormName	pbFormName and cbFormName describe a character string containing the name of the form to be read. If cbFormName is 0, the form name is the file name minus the file name extension, if there is one.
pbRgValuesRet cbMax	pbRgValuesRet and cbMax describe a user-supplied buffer. The ExtractValues routine returns information in this buffer.
	Its structure is identical to the input buffer structure for the AddValues routine.
pcbValuesRet	pcbValuesRet describes a word returned by the routine, containing the actual count of valid bytes in the returned buffer.

FonctForm

FonctForm allows the application program to dynamically redefine a function key as:

- □ an invalid function key
- □ a form validation function key
- □ a form cancel function key

Since the modification is done only in memory, it is lost when the application program exits.

The procedural interface is:

FonctForm (pbFonct, cbFonct): ErcType

Refer to table 4-6 for names and descriptions of FonctForm procedural interface parameters.

Table 4-6 **FonctForm Procedural Interface Parameters**

Name	Description
pbFonct cbFonct	pbFonct and cbFonct describe a memory buffer containing the new key values. This buffer must be an array of 2-byte elements, one for each key to redefine: □ First byte: ASCII code of the key, in encoded mode □ Second byte: New value: 00 = Illegal 01 = Form validation 02 = Form cancel

GetFieldInfo

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

The procedural interface is:

GetFieldInfo (pForm, pbFieldName, cbFieldName, index, pFieldInfoRet, cbFieldInfoMax): ErcType

Refer to table 4-7 for names and descriptions of GetFieldInfo procedural interface parameters. Refer to table 4-8 for information returned by GetFieldInfo.

Table 4-7 **GetFieldInfo Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
pbFieldName cbFieldName	pbFieldName and cbFieldName describe a character string naming the field to be interrogated. In matching field names, GetFieldInfo ignores the distinction between uppercase and lowercase.
index	index specifies the field of a repeating field to be interrogated. GetFieldInfo ignores this parameter, if the named field is not repeating.
pFieldInfoRet cbFieldInfoMax	pFieldInfoRet and cbFieldInfoMax decribe the memory work area into which GetFieldInfo returns the field information. Table 4-8 outlines the format of this information.

Table 4-8 **Information Returned by GetFieldInfo**

Offset	Field	Size (bytes)	Description
0	iCol	2	a column number relative to the left edge of the form
2	iLine	2	a line number relative to the top of the form
4	cCol	2	the width of the field in the column
6	fShowDefault	2	TRUE if the show-default property is turned on
8	fAutoExit	2	TRUE if the auto-exit property is turned on
10	fRepeating	2	TRUE if the repeating property is turned on
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 lower bound of indexes for a repeating field
18	indexLast	2	the upper bound of indexes for a repeating field
20	fMandatory	2	TRUE if the field is mandatory
22	fProtected	2	TRUE if the field is protected
24	fSecret	2	TRUE if the secret property is turned on
26	Justif	1	an encoding of the justification: 01: justified on decimal mark 02: right-justified 04: left-justified
27	DecimalMax	1	the maximum number of decimals for a decimal field
28	Control	1	an encoding of the data type control: 01: text 02: alphanumeric 04: alphabetic 08: numeric

Table 4-8 **Information Returned by GetFieldInfo** (continued)

Offset	Field	Size (bytes)	Description
29	SeqNumber	1	the sequence number of field
30	cchDefault	2	the length of the default value
32	rgchDefault	132	the default value
164	sbValProc	31	the name of the user validation routine (first byte is the length)
195	sbHelpMsg	61	the Help message (first byte is the length)
256	rgsbList Values	488	the list of values consists of an array of up to eight character strings. The first byte of each element indicates string length. The end of the list is indicated by a zero count if there are fewer than eight strings.

LockKbd

LockKbd requires that the user press **CANCEL** before using the keyboard. When a program calls LockKbd, the system beeps and the keyboard locks; further input is refused until the user presses **CANCEL.**

Whenever the user presses a key other than **CANCEL**, the system beeps and does not respond.

The procedural interface is:

LockKbd: ErcType

OpenForm

OpenForm reads information from an open-file form into a work area in memory. Once open, the form work area is self-contained; OpenForm can then close the file.

The procedural interface is:

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

Refer to table 4-9 for names and descriptions of OpenForm procedural interface parameters.

Table 4-9 **OpenForm Procedural Interface Parameters**

Name	Description
fh	fh is an open file handle for the form file (or library file).
pbFormName cbFormName	pbFormName and cbFormName describe a character string containing the form name of the form to be read. In matching form names, OpenForm ignores the distinction between uppercase and lowercase.
pFormRet cbMax	pFormRet and cbMax describe the memory work area into which the open form is returned. 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).

OutForm

OutForm returns information about a form and the different fields contained in this form. If the form is not already open, the OutForm operation itself can load it into the memory work area.

The procedural interface is:

OutForm (pForm, cbMaxForm, pbFileName, cbFileName, pbFormName, cbFormName, pbRet, cbMax, pcbRet): ErcType

Refer to table 4-10 for names and descriptions of OutForm procedural interface parameters. Refer to table 4-11 for information returned by OutForm.

Table 4-10 **OutForm Procedural Interface Parameters**

Name	Description
pForm cbMaxForm	pForm and cbMaxForm describe the work area of an already open form, or the work area into which OutForm must load the form.
pbFileName cbFileName	pbFileName and cbFileName describe a character string containing the name of the form file (or library file). If cbFileName is 0, no disk file is accessed; OutForm is supposed to have already loaded the form into the work area.
pbFormName cbFormName	pbFormName and cbFormName describe a character string containing the form name of the form to be read. OutForm ignores these parameters, if it is not required to load the form from disk into the work area. If cbFormName is 0, the form name is the file name minus the file name extension, if there is one.
pbRet cbMax	pbRet and cbMax describe the memory buffer into which OutForm returns the information.
pcbRet	pcbRet is a pointer to a word into which OutForm copies the count of bytes of significant information returned.

Structure of the returned buffer:

The returned buffer consists of a header followed by one field description block for each field of the form. Table 4-11 outlines the format of this information.

Table 4-11 **Information Returned by OutForm**

Offset	Field	Size (bytes)	Description
Header (general information about the form):			
0	sbFormName	N	form name (first byte is the size)
N	cField	2	number of fields
N+2	cLine	1	number of lines
Field description block (this field description block is repeated cField times):			
0	sbFieldName	n	field name (first byte is the size)
n	cCol	1	width of the field (number of columns)
n+1	bType	1	field type: Bit 0 = 1: Text Bit 1 = 1: Alphanumeric Bit 2 = 1: Alphabetic Bit 3 = 1: Numeric Bits 4 to 7: reserved
n+2	bJust	1	justification: Bits 0 to 3: number of decimals if the field is justified on decimal mark Bit 4 = 1: Justified on decimal mark Bit 5 = 1: Right-justified Bit 6 = 1: Left-justified Bit 7: reserved
n+3	wFlags	2	flags: Bits 0 to 3: Selected video attribute Bits 4 to 7: Unselected video attribute Bit 8 = 1: Secret Bit 9 = 1: Mandatory Bit 10 = 1: Protected Bits 11 and 12: reserved Bit 13 = 1: Repeating Bit 14 = 1: Auto-exit Bit 15 = 1: Show default

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 effect on ReadField. ReadField returns the default value, whether or not the default is displayed, unless a user entry changes the field's value.

The procedural interface is:

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

Refer to table 4-12 for names and descriptions of ReadField procedural interface parameters.

Table 4-12 **ReadField Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
pbFieldName cbFieldName	pbFieldName and cbFieldName describe a character string naming the field to be read. In matching field names, ReadField ignores the distinction between uppercase and lowercase.
index	index specifies the field of a repeating field to be read. ReadField ignores this parameter, if the named field is not repeating.
pbRet cbMax	pbRet and cbMax describe the memory work area into which ReadField returns the data.
pcbRet	pcbRet is the memory address of the word into which ReadField returns the count of bytes read.
pType	pType is the memory address of a type code ReadField uses in encoding the data.

RestoreForm

RestoreForm restores a previously displayed form to its default values, or to values the application program supplies in an initialization buffer. RestoreForm always performs this operation without accessing any file.

The procedural interface is:

RestoreForm (pForm, pbInit, cbInit): ErcType

Refer to table 4-13 for names and descriptions of RestoreForm procedural interface parameters.

Table 4-13 **RestoreForm Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
pbInit cbInit	pbInit and cbInit describe a buffer that contains the initialization values for the different fields of the form. If cbInit is 0, RestoreForm restores the default values. The RestoreForm buffer structure is the same as for the EditForm operation.

SetFieldAttrs

SetFieldAttrs sets the field character attributes.

The procedural interface is:

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

Refer to table 4-14 for names and descriptions of SetFieldAttrs procedural interface parameters.

Table 4-14 **SetFieldAttrs Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
pbFieldName cbFieldName	pbFieldName and cbFieldName describe a character string naming the field to be modified. In matching field names, SetFieldAttrs ignores the distinction between uppercase and lowercase.
index	index specifies the field of a repeating field to be modified. SetFieldAttrs ignores this parameter, if the named field is not repeating.
attr	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 through P (refer to table 3-3, Character Attributes, in section 3) □ 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

SetFieldType

SetFieldType allows the application program to dynamically redefine the type (optional, mandatory, or protected) of a specified field, or of all the fields of the form.

The procedural interface is:

SetFieldType (pForm, fAll, pbFieldName, cbFieldName, index, type): ErcType

Refer to table 4-15 for names and descriptions of SetFieldType procedural interface parameters.

Table 4-15 **SetFieldType Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
fAll	fAll is TRUE or FALSE. If fAll is TRUE, type redefinition applies to all the fields of the form; SetFieldType ignores pbFieldName, cbFieldName and index. If fAll is FALSE, type modification applies only to the field specified by pbFieldName, cbFieldName and index.
pbFieldName cbFieldName	pbFieldName and cbFieldName describe a character string naming the field to be redefined.
index	index specifies which instance of a repeating field is to be redefined. SetFieldType ignores this parameter, if the named field is not repeating.
type	type is a word containing any of the following: 0: Optional 1: Mandatory 2: Protected

StoreFieldData

The StoreFieldData operation extracts the value of a specified field from a buffer returned by the TypeForm operation, and moves the value into a work area in application program memory. It encodes the extracted data according to any type code (for example, **Character** or **Binary**).

The procedural interface is:

StoreFieldData (pForm, pbFieldName, cbFieldName, index, pbBuf, pbRet, cbMax, pInfoRet, pType): ErcType

Refer to table 4-16 for names and descriptions of StoreFieldData procedural interface parameters.

Table 4-16 **StoreFieldData Procedural Interface Parameters**

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>pForm</td><td>pForm is a pointer to the work area of an open form.</td></tr>
<tr><td>pbFieldName
cbFieldName</td><td>pbFieldName and cbFieldName describe a character string naming the field whose value is to be extracted.</td></tr>
<tr><td>index</td><td>index specifies from what instance of a repeating field the value is to be extracted. StoreFieldData ignores this parameter, if the named field is not repeating.</td></tr>
<tr><td>pbBuf</td><td>pbBuf points to a buffer returned by a previous TypeForm operation.</td></tr>
<tr><td>pbRet
cbMax</td><td>pbRet and cbMax describe the memory work area into which StoreFieldData is to move the extracted data.</td></tr>
<tr><td>pInfoRet</td><td>pInfoRet points to a memory area into which StoreFieldData returns the following information:
<table>
<tr><th></th><th></th><th>Size</th><th></th></tr>
<tr><th>Offset</th><th>Field</th><th>(bytes)</th><th>Desription</th></tr>
<tr><td>0</td><td>cbRet</td><td>2</td><td>size (in bytes) of significant data moved to the memory area pointed by pbRet</td></tr>
</table>
</td></tr>
</table>

Table 4-16 **StoreFieldData Procedural Interface Parameters** (continued)

Name	Description			
	Offset	Field	Size (bytes)	Description
	2	State	2	field state (equivalent to bState byte in the buffer returned by TypeForm)
pType	pType is the memory address of a type code StoreFieldData uses in encoding the data.			

TypeForm

The application program calls TypeForm to allow the user to interactively modify the contents of a previously displayed form. The application program initially specifies the first field to be selected for input. TypeForm then automatically selects other fields, depending on the field exit conditions it encounters (**Next field**, **Previous field**, **First field**, or **Last field** function keys, and auto-exit property of the field). It skips all protected fields. If the user presses the **Next field** key when the currently selected field is the last of the form, or the Previous field key when it is the first, the field remains selected and the system beeps to notify the user of the error. The **First field** or **Last field** function key enables a user to jump, respectively, to the first or last field of the form.

The TypeForm operation returns control to the application program only when the user presses the **Form validation** or **Form cancel** function key. However, TypeForm can automatically call a user validation routine for any individual field. The user must have declared the name of this routine when defining the field; TypeForm calls it each time the user exits from the field after having modified its contents, if the field has not become empty, and if the user has not exited through a **Form cancel** function key. (Refer to User Validation Routines in this section for additional information.)

If the user has validated the input and has left no mandatory field empty, the TypeForm operation returns to the application program all the values of the different fields grouped together in a single buffer. If any mandatory field is empty when the user presses the **Form validation** function key, the user is notified of the error and asked to fill in the first empty mandatory field. The type code that TypeForm uses in encoding the data is always **Character**. If the user has cancelled the input, TypeForm returns an appropriate error code. Data the user enters is not passed to the application program.

The procedural interface is:

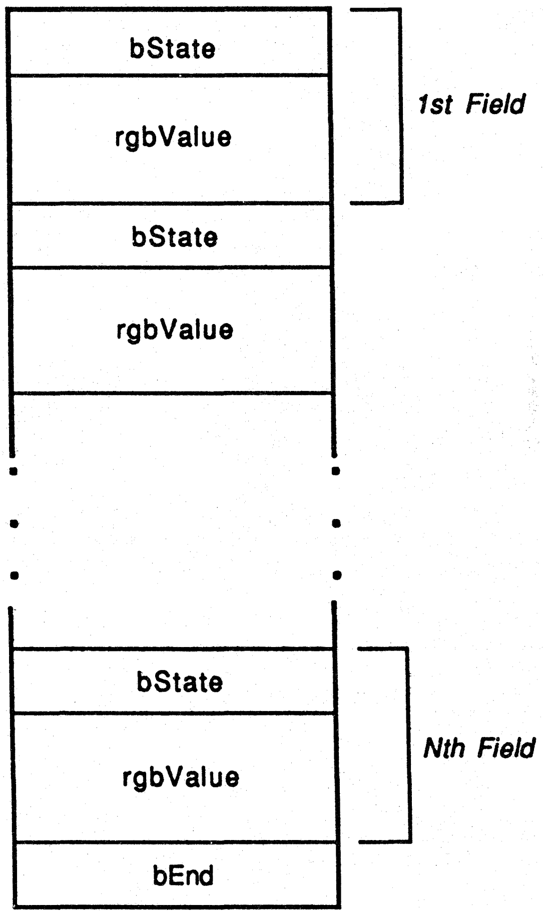
TypeForm (pForm, pbRet, cbMax, pbFirstField, cbFirstField, index, pExitStateRet): ErcType

Refer to table 4-17 for names and descriptions of TypeForm procedural interface parameters.

Table 4-17 **TypeForm Procedural Interface Parameters**

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>pForm</td><td>pForm is a pointer to the work area of an open form.</td></tr>
<tr><td>pbRet
cbMax</td><td>pbRet and cbMax describe the memory buffer into which TypeForm returns the data.</td></tr>
<tr><td>pbFirstField
cbFirstField</td><td>pbFirstField and cbFirstField describe a character string naming the field to be initially selected for input. If cbFirstField is 0, TypeForm selects the first field of the form.</td></tr>
<tr><td>index</td><td>index specifies which instance of a repeating field is to be initially selected for input. TypeForm ignores this parameter if cbFirstField is 0, or if the named field is not repeating.</td></tr>
<tr><td>pExitStateRet</td><td>pExitStateRet points to a 4-byte memory area into which TypeForm returns the following information:
<table>
<tr><th></th><th></th><th>Size</th><th></th></tr>
<tr><th>Offset</th><th>Field</th><th>(bytes)</th><th>Description</th></tr>
<tr><td>0</td><td>fill</td><td>1</td><td>unused</td></tr>
<tr><td>1</td><td>ch</td><td>1</td><td>the terminating character code</td></tr>
<tr><td>2</td><td>cbRet</td><td>2</td><td>size (in bytes) of significant data returned in the buffer</td></tr>
</table>
The buffer's structure must take the form shown in figure 4-3.</td></tr>
</table>

Figure 4-3 **TypeForm Buffer Structure**



In figure 4-3:

bState is a byte that can have any of the following values:

00 =	field not empty and not modified
01 =	field empty and not modified
02 =	field not empty and modified
03 =	field empty and modified

There must be a bState byte for each field of the form. If bState is 01 or 03, the bState byte for the next field must immediately follow it.

rgbValue is the value TypeForm reads from the field, and which it returns as a string of characters.

TypeForm never returns binary zeroes. It always returns ASCII spaces (20h), except if the field is empty (no other character than space or binary zero). It always returns non-significant ASCII zeroes (30h). If the field is decimal, it includes the decimal mark in the returned value, except if the decimal part is empty.

rgbValue must immediately follow a bState byte containing 00 or 02.

bEnd is a byte that marks the end of the buffer. It is set to 00.

If a user has defined a specific sequence for this form, the elements (bState[,rgbValue]) for the different fields of the form must appear in the buffer in the order of the sequence numbers; if a user has not defined a sequence, the elements must appear according to the location of the fields on the screen (from top to bottom and from left to right).

Note: This structure is compatible with the structure of the initialization buffer used by the EditForm operation. The buffer returned by TypeForm, therefore, can also serve as input for EditForm.

UndisplayForm

UndisplayForm removes a form from the screen, and replaces it with the null character (code 0h). The contents of any fields that were not read by the application program are lost.

The procedural interface is:

UndisplayForm (pForm): ErcType

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

UserFillField

UserFillField highlights the field with the selected character attributes and sets the cursor in the field.

If the user specified auto-exit when defining the field, UserFillField removes the highlight from the field and restores the unselected character attributes when the user enters data in the last character cell.

UserFillField passes the terminating keystroke back to the calling program in exitState.ch. The program interprets the keystroke (for example, the user may use **NEXT** to sequence through fields).

UserFillField does not include the terminating key value in the data; if the user, however, enters a character in the last position of an auto-exit field, UserFillField includes the character as data, sets exitState.ch to right arrow (code 12h), and sets exitState.fAuto-Exit to TRUE. Likewise, if the user presses the **Left Arrow** key or the **Backspace** key while the cursor is in the first position of a field, the code corresponding to the key (0eh and 08h, respectively) is returned in exitState.ch and exitState.fAuto-Exit is set to TRUE.

If the user tries to move the cursor or enter characters beyond the field boundary, the system beeps, with the following exception: With the repeating fields characteristic, text can be entered into a field and continued by moving to the next field if both are repeating fields.

If the user presses **CODE-Left Arrow**, the cursor moves to the left edge of the field; if the user presses **CODE-Right Arrow**, the cursor moves to the end of the field.

The user enters or edits text in either insert or overtype mode, and can press **CODE-DELETE** or **CODE-BACKSPACE** to delete the contents of the entire field.

The procedural interface is:

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

Refer to table 4-18 for names and descriptions of UserFillField procedural interface parameters. Refer to table 4-19 for information returned by UserFillField.

Table 4-18 **UserFillField Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
pbFieldName cbFieldName	pbFieldName and cbFieldName describe a character string naming the field to be filled. In matching field names, UserFillField ignores the distinction between uppercase and lowercase.
index	index specifies the field that is to be completed (for a repeating field). UserFillField ignores this parameter, if the field is not repeating.
pInitState	pInitState points to a descriptor block.
pExitStateRet	pExitStateRet points to a memory area. Table 4-19 gives further information on these two parameters.

Table 4-19 **Information Returned by UserFillField**

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 user changed the field contents
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.

The procedural interface is:

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

Refer to table 4-20 for names and descriptions of WriteField procedural interface parameters.

Table 4-20 **WriteField Procedural Interface Parameters**

Name	Description
pForm	pForm is a pointer to the work area of an open form.
pbFieldName cbFieldName	pbFieldName and cbFieldName describe a character string naming the field to be written. In matching field names, WriteField ignores the distinction between uppercase and lowercase.
index	index specifies which field of a repeating field is to be written. WriteField ignores this paramenter, if the named field is not repeating.
pb cb	pb and cb describe a memory area containing the data to be displayed.
pType	pType is the memory address of a type code WriteField uses in decoding the data.

User Validation Routines

A user validation routine must be a procedure defined with the PUBLIC attribute and linked with the application program. When configuring Forms Run-Time, you must declare the names of all user validation routines that are to be used. You can do this by editing the assembler module FmValProc.asm. Information about how to modify this module is included as comments in the module itself. The user then must assemble FmValProc.asm and store, using the Librarian utility, the resulting object module in Forms.lib.

A user validation routine must be a Boolean function that returns TRUE if it accepts the current field value as valid, FALSE if it does not. Its interface must be as follows:

ValProc (pFvb): Boolean

pFvb is a pointer to a Field Validation Block, passed by TypeForm to the user validation routine, with the following structure:

Offset	Field	Size (bytes)
0	sbFieldName	41
41	index	2
43	iBlank	2
45	sbValue	133
178	ich	2
180	psbErrorMsg	4

The Field Validation Block is used in both directions, for passing information from TypeForm to the user validation routine (input), and for returning information from the user validation routine to TypeForm (output).

The meanings of the various parameters in the Field Validation Block are as follows:

□ sbFieldName / index / iBlank:

Input: Name, index and number of the field to be validated.

sbFieldName is an array of bytes, where the first byte is the length of the field name.

iBlank is the internal number of the field. It is equal to the sequence number minus 1, and therefore it starts at 0.

Output: TypeForm ignores these parameters if the value of the current field is not validated. If it is, you can employ these parameters to specify what is the next field to select for input.

There are three cases:

- □ You want to let TypeForm choose the next field, depending on the keystroke that caused the exit from the current field.

  You must set:

  sbFieldName(0) = 0;

  iBlank = 0FFFFh.
- □ You want to select the next field by its internal number.

  You must set:

  sbFieldName(0) = 0;

  iBlank = desired number.
- □ You want to select the next field by its name and index.

  You must set:

  sbFieldName and index to the desired values;

  iBlank to any value, as it is ignored if

  sbFieldName(0) is not equal to 0.

  If the user has exited the current field through a **Form validation** function key, Forms Run-Time processes this key anyway, regardless of the values returned for sbFieldName, index and iBlank.
- □ sbValue:

  Input: Current value of the field. sbValue is an array of bytes, where the first byte is the length of the value.

  Output: Ignored.

- ich:

  Input: The final cursor position in the field.

  Output: TypeForm ignores the ich parameter if the current value is validated. If not validated, it must contain the initial location at which the cursor should be repositioned in the field.

- psbErrorMsg:

  Input: Non-significant.

  Output: TypeForm ignores the psbErrorMsg parameter if the current value is validated. If not validated, it must contain the address of the user-specific error message to be displayed.

psbErrorMsg is a pointer to an array of bytes, where the first byte is the length of the error message. If the length is equal to 0, Forms displays the standard message **Invalid data**.

Forms Run-Time Customization

You can customize these four modules:

- FmTabKey.asm

  a keyboard definition table contained in an assembly language source file, FmTabKey.asm for Forms Designer version 4.0 functionality.

- FmTabKey5.asm

  a keyboard definition table contained in an assembly language source file, FmTabKey5.asm for Forms Designer version 5.0 functionality.

- FmRgtd.asm

  a master table contained in an assembly language source file, FmRgtd.asm

- FmValProc.asm

  a user validation procedure table containing an assembly language source file FmValProc.asm

The result of assembling the .asm files is an object file. You use the BTOS Linker to link these object files with the other object modules of the application program and the Forms Run-Time library to produce your application program. (Refer to the *BTOS II Language Development Linker and Librarian Programming Guide*.)

Keyboard Configuration

The Forms.lib library contains the TabKey table, which is an independent object module obtained by assembling the desired FmTabKey.asm file. The TabKey table contains 256 control bytes, which define keyboard functions; each key on the keyboard corresponds to a control byte contained in the TabKey table.

You can reconfigure (or redefine) some of the keys on the keyboard to perform functions other than the standard ones they normally perform. Table 4-21 shows the functions that you can reassign to other keys and the standard keys that currently provide those functions.

Table 4-21 **Keys That Can Be Reconfigured**

Function	**Default Key**
Form Validation	**GO**
Form Cancel	**CANCEL**
Next Field	**RETURN, NEXT, TAB, Down Arrow**
Previous Field	**Up Arrow**
First Field	**CODE-Up Arrow**
Last Field	**CODE-Down Arrow**
Next Value	**SCROLL UP**
Previous Value	**SCROLL DOWN**
Decimal Mark*	**Decimal Point (.)**

* You can redefine the Decimal Mark key, using a special parameter (chDecimalMark) in the FmTabKey.asm or FmTabKey5.asm module, to be either a period (.) or a comma (,). For more information, refer to Forms Run-Time Customization.

The following keys are reserved and cannot be reassigned:

- cursor-movement keys (**Left Arrow, Right Arrow, CODE-Left Arrow**, and **CODE-Right Arrow**)
- the **HELP** key
- the literal-insert key combination (**CODE-=**)
- function keys **F1** through **F10**

There are two ways to reconfigure the keys listed in table 4-21:

- by modifying the FmTabKey.asm or FmTabKey5.asm assembler module that contains the TabKey table; then assembling FmTabKey.asm or FmTabKey5.asm and using the BTOS Librarian to store the object module. (Refer to the *BTOS II Language Development Linker and Librarian Programming Guide*.)
- by using the FonctForm operation. (Refer to Forms Run-Time Services.)

Some keys are data keys; their control byte indicates whether they are acceptable as alphabetic, numeric, alphanumeric, or text characters. You cannot redefine characters that have ASCII codes 00h through 03h as data characters.

Note: Numbers are decimal except when suffixed with **h** (hexadecimal); for example, 10h = 16 and 0FFh = 255.

Be aware that any keys whose functions you change by modifying the TabKey table cannot be used when testing forms with the Form Editor's Test Drive function (described in section 3); the Form Editor uses its own TabKey table, which is based on a standard keyboard configuration.

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.

Assembly language source code generates the table. 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.

Type Codes

A type code is presented to Forms Run-Time whenever it reads or writes a field; 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, a colon and a string of decimal digits precede the final period. Examples of type codes are **Binary** and **Character:50**.

You can configure the type system when you employ user-supplied type codes and conversion routines to link Forms Run-Time and the calling program.

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.

Adding Types

When a ReadField or WriteField requires type conversion, the rgtd table is searched for a type code matching the one supplied to the ReadField or WriteField in progress. In matching type codes, ReadField or WriteField ignores the distinction between uppercase and lowercase, and also the numeric suffix of the type code (if any). Thus the type **Character**, defined above, matches all of the following type codes:

CHARACTER:50
Character:100
cHarAcTer:1

If ReadField or WriteField finds no match for a given type code, it returns the message **Bad type specification**. If it finds a match in the table, ReadField calls the associated EncodeProc, or WriteField calls the associated DecodeProc, to perform the conversion.

EncodeProc

EncodeProc (used by ReadField) must have the interface:

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

Refer to table 4-22 for names and descriptions of EncodeProc parameters.

Table 4-22 **EncodeProc Parameters**

Name	Description
prgch	prgch is a pointer to the text Forms Run-Time reads from a field.
cch	cch is the length of the text Forms Run-Time reads from a field.
pb cb	pb and cb describe the data area passed to ReadField.
pcbRet	pcbRet is a pointer to a word to be set to the number of encoded bytes placed by EncodeProc into pb.
sType	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), EncodeProc returns the message **Invalid data**; otherwise, it returns the message **OK**.

DecodeProc

DecodeProc (used by WriteField) must have the interface:

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

Refer to table 4-23 for names and descriptions of DecodeProc parameters.

Table 4-23 **DecodeProc Parameters**

Name	Description
prgch	prgch is a pointer to the area in which DecodeProc places text to be written to a field.
cch	cch is the length of an area in which DecodeProc places text to be written to a field.
pb cb	pb and cb describe the existing encoded data.
pcbRet	pcbRet is a pointer to a word to be set to the number of decoded bytes.
sType	sType is the binary value of the type code's numeric suffix (if any).

Example:

To implement a new type called Money, use the following procedure:

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 EncodeMoney reads a string containing other characters or a string in a different format, it 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 application program is responsible for range checking and other data validation of encoded data. The BASIC program in appendix B checks the PartNumber field for range validity in lines 720-740.

Forms Run-Time 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 approaches are:

- You can set the program to signal an error by beeping and repositioning the cursor in the field in which invalid data was entered, or in the field where the cursor appeared when the user pressed an erroneous key.

  The interface to UserFillField permits you to specify the character position in the field where the cursor is initially positioned.

  The interface to TypeForm permits the cursor to move to the first field or last field in a form; you can use this to respond to an erroneous keystroke. Typical erroneous keystrokes include pressing the Next Field key when the cursor is in the last field, and pressing the Previous Field key when the cursor is in the first field.

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

  LockKbd beeps for each character the user enters, and disregards any entry after the erroneous entry until the user presses **CANCEL**. 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 that the user realizes an error has occurred.

Forms Structure

This section describes the representation of a form as it resides in memory and as it resides on a BTOS disk file.

This section provides you with descriptions of the following:

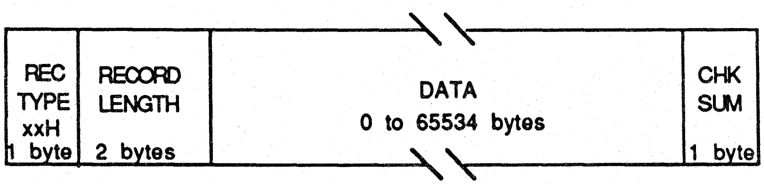
- the external representation of a form as it resides on a BTOS disk file
- the internal representation of a BTOS Forms Designer release 6.0 form as it resides in memory after having been linked to an application program through the BTOS Linker or loaded into memory from a disk file using OpenForm.

External Structure

The BTOS disk file is of a format compatible with BTOS object modules so that a form can be linked to an application using the BTOS Linker. You can use BTOS Librarian to combine multiple forms into a single library file.

The BTOS object format consists of a series of tables of the same general format as shown in figure 5-1.

Figure 5-1 **Form Format - External Representation**



REC TYP is the first byte in each record and contains a value between 0 and 255 decimal, which indicates the record type.

RECORD LENGTH is the second field in each record. It contains the number of bytes in the record, excluding the first two fields. This field is two bytes, using the byte-reversed format, which is used for all two-byte fields in all records.

DATA is a series of bytes, from 0 to 65534 decimal in number. The content and size of the data depends on the data type.

CHK SUM is the last field of each record, and contains a check sum containing the twos complement of the sum (modulo 256 decimal) of all other bytes in the record. Therefore, the sum (modulo 256 decimal) of all bytes in the record equals 0.

Using the format shown in figure 5-1, each form is represented by a series of at least seven tables. The following numeric values are hexadecimal unless otherwise noted:

- **Table 1, Rec Typ = 80**

  The data for this table is the name of the form in ASCII. The first byte of the name string is the length (in bytes) of the remainder of the string. This name is usually the name of the form file without the .form suffix.

- **Table 2, Rec Typ = 88**

  The data for this table is a comment identifying the level of forms object format supported. Refer to the following listings for the exact format of this comment.

- **Table 3, Rec Typ = 96**

  The data contained in this table is the ASCII string DATA, preceded by the size of the string (4).

- **Table 4, Rec Typ = 98**

  The first byte of data in this table is 48. The second field is a two-byte size of the actual form data as it is represented in memory for usage by the Forms Run-Time Library (refer to Internal Structure in this section). The final three bytes are 1, 1, 8.

- **Table 5, Rec Typ = 90**

  This table describes the public name by which the form data may be referenced. This name is normally the same as the form name used in table 0. The format of the table is: 1, 1, size of the name string in bytes, the ASCII string, 0, 0, 9

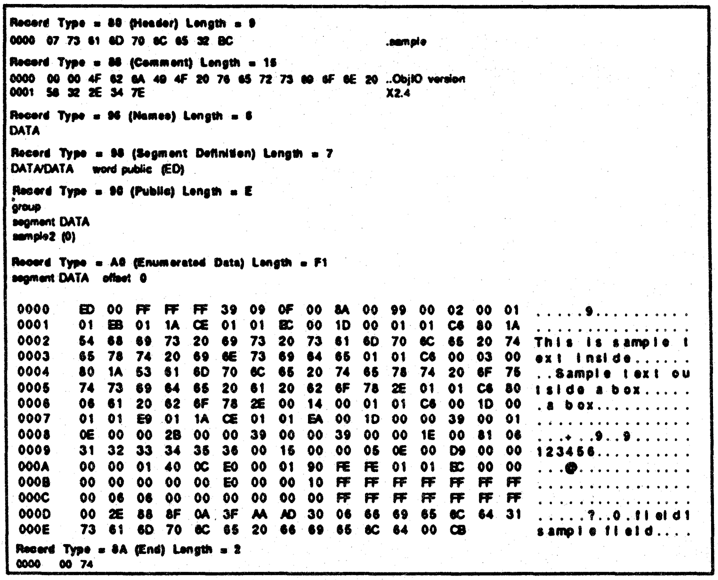
- □ **Table 6 through n-1, Rec Typ = A0**

  These tables contain the actual form data in memory resident format. Each table contains up to 400 (1024 decimal) bytes of form data preceded by two fields. The first single-byte field contains 1. The next two-byte field contains the offset of the following data from the beginning of the form. Thus a form 810 bytes long would have three tables of sizes 400, 400, and 10 (plus three bytes each of header). The respective offsets would be 0, 400, and 800. The total size of the form data may be as large as 2000 bytes, the limit imposed by the FORMS internal format.

- □ **Table n, Rec Typ = 8A**

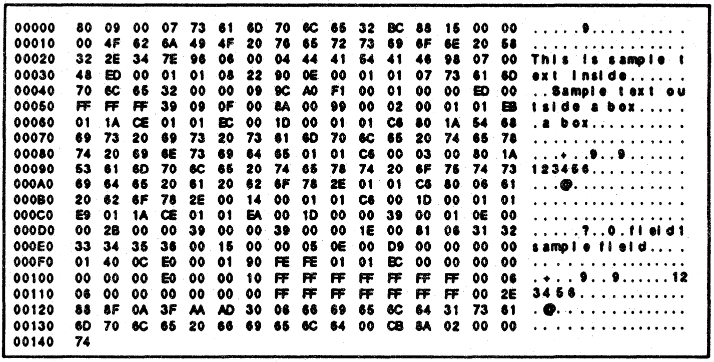
  This table signifies the end of the data. It contains a single-byte zero value.

The following listing displays a sample form in interpreted format:



```
Record  Type  =  80  (Header)  Length  =  9
0000  07 73 61 6D 70 6C 65 32 BC                                 .sample
Record  Type  =  88  (Comment)  Length  =  15
0000  00 00 4F 62 6A 49 4F 20 76 65 72 73 69 6F 6E 20  ..ObjIO version
0001  58 32 2E 34 7E                                     X2.4
Record  Type  =  96  (Names)  Length  =  6
DATA
Record  Type  =  98  (Segment  Definition)  Length  =  7
DATA/DATA    word public  (ED)
Record  Type  =  90  (Public)  Length  =  E
group
segment DATA
sample2 (0)
Record  Type  =  A0  (Enumerated  Data)  Length  =  F1
segment DATA   offset  0

 0000    ED  00  FF  FF  FF  39  09  0F  00  8A  00  99  00  02  00  01   .....9..........
 0001    01  EB  01  1A  CE  01  01  EC  00  1D  00  01  01  C6  80  1A   ................
 0002    54  68  69  73  20  69  73  20  73  61  6D  70  6C  65  20  74   This is sample t
 0003    65  78  74  20  69  6E  73  69  64  65  01  01  C6  00  03  00   ext inside......
 0004    80  1A  53  61  6D  70  6C  65  20  74  65  78  74  20  6F  75   ..Sample text ou
 0005    74  73  69  64  65  20  61  20  62  6F  78  2E  01  01  C6  80   tside a box.....
 0006    06  61  20  62  6F  78  2E  00  14  00  01  01  C6  00  1D  00   .a box..........
 0007    01  01  E9  01  1A  CE  01  01  EA  00  1D  00  00  39  00  01   ................
 0008    0E  00  00  2B  00  00  39  00  00  39  00  00  1E  00  81  06   ...+..9..9......
 0009    31  32  33  34  35  36  00  16  00  00  05  0E  00  D9  00  00   123456..........
 000A    00  00  01  40  0C  E0  00  01  90  FE  FE  01  01  EC  00  00   ...@............
 000B    00  00  00  00  00  E0  00  00  10  FF  FF  FF  FF  FF  FF  FF   ................
 000C    00  06  06  00  00  00  00  00  00  FF  FF  FF  FF  FF  FF  FF   ................
 000D    00  2E  88  8F  0A  3F  AA  AD  30  06  66  69  65  6C  64  31   .....?..0.field1
 000E    73  61  6D  70  6C  65  20  66  69  65  6C  64  00  CB            sample field....
Record  Type  =  8A  (End)  Length  =  2
0000   00 74
```

The following listing shows the same form file in raw format:



```
00000   80 09 00 07 73 61 6D 70 6C 65 32 BC 88 15 00 00   .....9..........
00010   00 4F 62 6A 49 4F 20 76 65 72 73 69 6F 6E 20 58   ................
00020   32 2E 34 7E 96 06 00 04 44 41 54 41 46 98 07 00   This is sample t
00030   48 ED 00 01 01 08 22 90 0E 00 01 01 07 73 61 6D   ext inside......
00040   70 6C 65 32 00 00 09 9C A0 F1 00 01 00 00 ED 00   ..Sample text ou
00050   FF FF FF 39 09 0F 00 8A 00 99 00 02 00 01 01 EB   tside a box.....
00060   01 1A CE 01 01 EC 00 1D 00 01 01 C6 80 1A 54 68   .a box..........
00070   69 73 20 69 73 20 73 61 6D 70 6C 65 20 74 65 78   ................
00080   74 20 69 6E 73 69 64 65 01 01 C6 00 03 00 80 1A   ...+..9..9......
00090   53 61 6D 70 6C 65 20 74 65 78 74 20 6F 75 74 73   123456..........
000A0   69 64 65 20 61 20 62 6F 78 2E 01 01 C6 80 06 61   ...@............
000B0   20 62 6F 78 2E 00 14 00 01 01 C6 00 1D 00 01 01   ................
000C0   E9 01 1A CE 01 01 EA 00 1D 00 00 39 00 01 0E 00   ................
000D0   00 2B 00 00 39 00 00 39 00 00 1E 00 81 06 31 32   .....?..0.field1
000E0   33 34 35 36 00 15 00 00 05 0E 00 D9 00 00 00 00   sample field....
000F0   01 40 0C E0 00 01 90 FE FE 01 01 BC 00 00 00 00   ................
00100   00 00 00 E0 00 00 10 FF FF FF FF FF FF FF 00 06   .+..9..9......12
00110   06 00 00 00 00 00 00 FF FF FF FF FF FF FF 00 2E   3456............
00120   88 8F 0A 3F AA AD 30 06 66 69 65 6C 64 31 73 61   .@..............
00130   6D 70 6C 65 20 66 69 65 6C 64 00 CB 8A 02 00 00   ................
00140   74
```
`

Internal Structure

The following describes the internal representation of a BTOS Forms Designer release 6.0 form as it resides in memory after having been linked to an application through the Linker facility or loaded into memory from a disk file using OpenForm. The format of the form data within a library or disk file is not addressed here. The following definitions are relative to Forms 6.0. Data definitions are represented as PL/M constructs. A byte occupies eight bits and a word equals sixteen bits.

Each form is a variable-sized structure composed of a fixed header and variant substructures of various type. The maximum total size of a form is 8192 bytes (8K). The topmost definition is:

DECLARE Form STRUCTURE

```
(
cb           WORD,
ifr          BYTE,
icol         BYTE,
iline        BYTE,
ccol         BYTE,
cline        BYTE,
bruns        WORD,
cbRuns       WORD,
brgblank     WORD,
iblankMac    WORD,
rgb(8177)    BYTE
);
```

Refer to table 5-1 for the names and descriptions of the Form STRUCTURE parameters.

Table 5-1 **Form STRUCTURE Parameters**

Name	Description
cb	the size of the entire form in bytes
ifr	the video frame in which the form is to be displayed
icol	the column within the video frame at which the upper left corner of the form is located
iline	the line within the video frame at which the upper left corner of the form is located
ccol	the maximum size of the form in columns
cline	the maximum size of the form in lines
bruns	the offset (from the start of the form) of the beginning of the runs (described later in this section) in the form
cbRuns	the size in bytes of all runs in the form
brgblank	the offset (from the start of the form) of the beginning of the area containing all blanks (described later in this section) in the form
iblankMac	the total number of blanks contained in the form
rgb	the array of bytes containing variable numbers and sizes of runs, blanks, and extended blanks

The form structure through iblankMac is fixed in size and meaning for all forms. The array rgb is a packed area containing the remainder of the data. The first class of data possible is the **run**. A **run** is a horizontal row across the video frame containing character data and attributes. The data types may be alphanumeric or graphical. Both are represented by members of the B20 family character set, with graphics being accomplished by combining characters represented by values above 0C0H. Each line within the form (**cline** lines) is represented by at least one **run**. The structure of each **run** is:

DECLARE Run STRUCTURE

```
(
tylooks        BYTE,
ccol           BYTE,
rgch (146)     BYTE
);
```

Refer to table 5-2 for the names and descriptions of the Run STRUCTURE parameters.

Table 5-2 **Run STRUCTURE Parameters**

Name	Description
tylooks	if bit 7 is 1, this is a "text" run described by ccol characters contained in first ccol elements of the rgch array. If bit 7 is 0, this is a "graphics" run of ccol identical characters of the value contained in the first element of the rgch array. In either case, the lower four bits of tylooks contain data regarding the attributes of the run. This constrains each run to describe a line of only a single attribute combination. The possible encodings are: Bit 0 - half bright Bit 1 - underlined Bit 2 - reverse video Bit 3 - blinking
ccol	the number of characters comprising this run
rgch	an array of 1 or more characters comprising the run

The next type of form data is the **blank**. Each field defined by the Forms Editor user is represented as a **blank**. Visually, this appears as a line of small boxes, and in fact, a **run** of these characters is generated for each **blank**. The data in the **blank** structure is fixed in format and represents the data entered into Forms Editor using **the Define Field** function (**F8**). The format is:

DECLARE Blank STRUCTURE

```
(
icolRel        BYTE,
ilineRel       BYTE,
ccol           BYTE,
iExtBlank      BYTE,
bsbName        WORD,
index          WORD,
ordering       BYTE,
control        BYTE,
justif         BYTE,
cchDefault     BYTE,
brgchDefault   WORD,
flagS          WORD
)';
```

Refer to table 5-3 for the names and descriptions of the Blank STRUCTURE parameter.

Table 5-3 **Blank STRUCTURE Parameters**

Name	Description
icolRel	the left most column of the blank relative to the first column of the form
ilineRel	the single line of the blank relative to the first line of the form
ccol	the number of columns the blank occupies
ExtBlank	the index of the extended blank associated with this blank (described later in this section)
bsbName	the offset from the beginning of the form of the string which represents the name of this blank. The first byte of the string is the number of bytes in the remainder.
index	the index used to identify repeating fields
ordering	the index used to define the order in which fields are selected for data entry by TypeForm if the default left to right and top to bottom order is not acceptable.

control	a flag which determines the format of the data to be contained within the blank. Values are:
	1 - text data;
	2 - alphanumeric data;
	3 - alphabetic data;
	4 - numeric data.

justif	an indicator of the size and justification of the data to be contained within the blank. Bits 0 through 3 contain the maximum number of decimal digits allowed for numeric entries. Bit 4 signifies decimal justification, bit 5 means right justified, and bit 6 indicates left justification.
cchDefault	the count of bytes in the string containing the default value of a blank
brgchDefault	the offset from the start of the blank of the character string defining the blanks default value

Table 5-3 **Blank STRUCTURE Parameters** (continued)

Name	**Description**

flagS	a series of flags indicating the state of various options. A one in the bit position means the option is valid. The following flag values are in **hexadecimal**:
	8000H - show default value;
	4000H - auto exit;
	2000H - repeating field;
	1000H - this blank has an associated extended blank (see below);
	0400H - prohibited field;
	0200H - mandatory field;
	0100H - secret field.
	The lower 8 bits contain the selected and unselected attributes of this blank. Bits 0 through 3 indicate the attributes to use in displaying this blank if unselected, and bits 4 through 7 are the selected attributes.

Each blank defined within a form may optionally have an extended blank associated with it. The extended blank contains data which is not necessarily specified for every blank. The extended blank area begins immediately following the packed blanks. The offset of this area from the start of the form may be computed as (brgblank + (iblankMac * blank size)). An extended blank is defined:

```
DECLARE ExtendedBlank STRUCTURE

                    (
                    bsbValProc          WORD,
                    bsbHelpMsg          WORD,
                    brgsbListValues     WORD,
                    iValueMax           BYTE,
                    iValue              BYTE,
                    rgbColors (8)       BYTE
                    );
```

Refer to table 5-4 for the names and descriptions of the ExtendedBlank STRUCTURE parameters.

Table 5-4 **ExtendedBlank STRUCTURE Parameters**

Name	Descripton
bsbValProc	the offset from the start of the form of the string containing the name of the user validation routine for the blank. The first byte of the string contains the number of bytes in the remainder.
bsbHelpMsg	the offset from the start of the form of the string containing the message to be displayed as if the HELP key is pressed while awaiting input from this blank. The first byte of the string contains the number of bytes in the remainder of the string.
brgsbListValues	the offset from the start of the form of a packed series of strings representing the possible values which may be contained in the blank. Each individual string begins with a byte count of the number of bytes in the remainder of the string. The actual string follows immediately after the byte count of the string.
iValueMax	the number of list values in the string series plus 1
iValue	the index of the default value among the list of values
rgbColors	for a normal extended blank, the first byte of this array contains the color index into the current palette (a value from 0 through 7) for the blank while selected. The second byte represents the color to be used if unselected. If a color palette has been selected for this form, there is a dedicated blank and extended blank used to represent the palette. The values of iColRel and iLineRel of this blank are set to 254, an area outside the frame. The extended blank associated with this special blank uses the rgbColors array to contain the encoding of the red, green, and blue components of the eight palette colors. Each component may have a value from 0 through 3 representing no intensity through full intensity for that particular component. These values are packed into 2 bits within each byte in the array. The masks for each color are: Red - 30H; Green - 0CH; Blue - 03H. In this fashion, six bits of each of the eight bytes construct the eight colors of the palette. Packed into the upper bit (bit 7) of the first three bytes of the array are the three bits necessary to select one of the eight colors of the palette to be used for the colors of the captions. Likewise, the fifth, sixth, and seventh bytes contain the color used for lines. This is shown in figure 5-2.

Figure 5-2 **Color Matrix**

Bit								
Color	7	6	5	4	3	2	1	0
0	Capt0		R1	R0	G1	G0	B1	B0
1	Capt1		R1	R0	G1	G0	B1	B0
2	Capt2		R1	R0	G1	G0	B1	B0
3			R1	R0	G1	G0	B1	B0
4	Line0		R1	R0	G1	G0	B1	B0
5	Line1		R1	R0	G1	G0	B1	B0
6	Line2		R1	R0	G1	G0	B1	B0
7			R1	R0	G1	G0	B1	B0

Following the extended region of the blank is space allocated to strings pointed to by the "bsb.." indices above. Some of these strings are: list of values; default values; and the name of the blank.

Error Messages

Error messages for Forms Designer are listed in the following three groups:

- Forms Run-Time error codes: 3700-3799
- Forms Editor error messages
- Forms Run-Time error messages

Forms Run-Time Error Codes

3700

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.

3701

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

Form too big

The work area supplied to OpenForm is too small to contain the named form. Use the Forms Reporter (FREPORT command) to determine the required work area.

3703

Form out of bounds

The coordinates passed to DisplayForm would result in a part of the form lying outside the frame. Use the Forms Reporter (FREPORT command) to determine the required height and width.

3704

Form not displayed

A Forms Run-Time service was called for a form that was not displayed. Display the form (use DisplayForm) before attempting any of the Forms Run-Time services.

3705

No such field

A Forms Run-Time service was called for a field (specified by pbFieldName, cbFieldName) and index that do not exist. Use the Forms Reporter (FREPORT command) to determine the correct name and index.

3706

Bad type specification

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.

3707

Bad data size

A ReadField or WriteField was attempted in which the cbMax or cb 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.

AddValues also returns this error message:

□ if the default value to be added is invalid, according to the data control type assigned to the field

□ if the data for a numeric field is not a valid number (for example, a numeric sign inside a number or a duplicated decimal point)

□ if the default value for a field defined with a list of values is not equal to the first characters of any values in the list

3708

Invalid data

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

AddValues also returns this error message if one or several default values are too large to fit in a field. As a result, AddValues truncates the user-supplied value name, processes all data in the buffer, and, as a warning, returns the last error code encountered.

3709-3739

Reserved.

3740

Protected field

UserFillField was called on a field defined as protected.

3741-3749

Reserved.

3750

Invalid initialization buffer

EditForm or RestoreForm was supplied with an initialization buffer that was not correctly formatted.

3751

Input cancelled

TypeForm was terminated by a Form cancel function key. No data is returned.

3752

Buffer too small

TypeForm or OutForm was supplied with a buffer too small for the size of the data to be returned.

3753

Invalid function key value

One or more of the function key values passed to FonctForm were invalid.

3754

No input field

TypeForm was called on a form containing only protected fields.

3755

No such validation routine

The name of the user validation routine associated with the field could not be found in the table generated by the FmValProc.asm module.

3756

Invalid Buffer

The value supplied to AddValues is inconsistent.

3757

Value List Too Large For Form

Including the user-supplied information will cause the Form descriptor to exceed its maximum size (8KB).

3758

Data Truncated In Returned Buffer

The user-supplied buffer is too small to contain all of the returned data. As a result, the system returns the maximum possible valid information and updates CbValuesRet.

3759-3799

Reserved.

Forms Editor Error Messages

A decimal-justified field must be a numeric

If you selected a field entry to be justified on the decimal point, you must specify the field to be a numeric (**Nu**) data type.

A picture is already displayed

You used **F3** to display a graphics picture while a graphics picture was already displayed.

Press **SHIFT-F3** to remove the current graphics picture before displaying another picture.

A protected field cannot be mandatory

You designated a protected field as a mandatory field. A field can be either protected or mandatory; it cannot be both.

Check whether you want to designate the field as protected or mandatory.

Bad file format

The system cannot find the file you specified in the READ FORM form. Check that you have specified the file correctly and try again.

The default volume and directory is the current path. To access a form in a different volume or directory, you must type in the complete file name using brackets.

Cannot move partial fields

You can MOVE or COPY a partial field only 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 system cannot find the file typed in a Read Form form. Check your file specification.

The current volume and directory is the default path.

To access a form in a different volume or directory, you must type in the complete file name using brackets.

Default inconsistent with list of values

You selected an invalid default value.

Select a default value from one of the values in the list.

Destination out of bounds

You cannot MOVE or COPY a field into an area that overlaps the display edge. Check the position both of your selection and of the cursor.

Discontinuous sequence numbers

You specified a fill-in sequence using numbers that were not sequential (continuous).

Specify sequence numbers sequentially, starting with 1 (for example, 1, 2, 3).

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.

Field too small to accept decimal justification

You defined a field for justification with the decimal point, but the field is too small to permit justification.

Create a field that is at least three columns wide.

File in use

The file you specified in the **[File]** field on the Text Insertion form is being edited from your workstation or from another workstation.

File is not a library

The file name you specified in the **[Library file]** field of the Graphics Display form does not exist in the library.

File name missing

On the Graphics Display form, you specified the name of a Library File that does not exist.

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 functions). 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 likely to occur if you have moved or copied fields.

Form is too wide

The Zoom function cannot reformat the display to 80 columns because the form does not fit. Use the View Edit Code function (press **CODE-V**) to check for any space characters that you can remove.

Graphics is not available on this workstation

You used **F3** to perform the graphics operation on a workstation that does not support graphics.

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.

Invalid data type

You specified an invalid data type for a field.

Identify the field as **Ab** (alphabetic), **Nu** (numeric), An (alphanumeric), or **Tx** (any ASCII characters).

Invalid display coordinates

You specified invalid column or line coordinates which would cause the form to appear outside the display area boundary.

Invalid file name

You specified an invalid Library file name on the Graphics Display form, or an invalid file name on the Text Insertion form.

Invalid justification

You specified an invalid justification selection for a field.

Identify the field as **L** (left justified), as **R** (right justified), or as a number between 1 and 15.

Last sequence number is greater than 200

Automatic incrementing of sequence numbers for a repeating field has caused sequence numbering to exceed 200. Sequence numbering cannot be greater than 200, which is the maximum number of fields in a form.

Multiple use of same field name

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

No fields defined

NEXT and Test Drive (F9) cannot be used unless you have defined a field.

No picture displayed

You used the Graphics Display form to display a graphics picture, but you entered the name of a bad library file or graphics module. This resulted in no graphics picture being displayed.

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 form does not exist in that file. If your file name is correct, check your current path.

Not enough memory for Graphics Display

There is not enough memory for you to use the graphics operation feature (**F3**). The Forms Editor dynamically allocates 64KB of memory for graphics operation.

Please enter a number in range 0-200

You specified a number greater than 200 for a fill-in sequence. A sequence number cannot exceed 200, which is the maximum number of fields in a form.

Please sequence all fields or none

You specified some form entries as part of a fill-in sequence and others as not part of that sequence.

If one field in the form has a nonzero sequence number, all fields must have a nonzero sequence number.

Selected area is too small

Too many lines or too much text appears in the field you selected with **MARK** and **BOUND**. The system truncates text that overlaps the field.

The file is not a Picture file

You entered the name of an invalid Picture file in the **[Graphics file]** field of the Graphics Display form.

The specified file is not a Word Processor file

The file you specified in the **[File]** field on the Text Insertion form is not a text file.

The file you specify must be a text file (that is, a file that is in the Word Processor or Forms Editor format).

Too many decimals

You specified too many decimals in a field.

The maximum number of decimals allowed is equal to the field's width minus 2.

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 functions). 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 likely to occur if you have moved or copied fields.

Value too large

The default value and/or listed values are so large that they exceed the width of the field.

Specify values that are not greater than the width of the field.

Forms Run-Time Error Messages

Alphabetic characters only

You tried to enter a nonalphabetic character in a field that the data control type defines as an alphabetic field.

Alphanumeric characters only

You tried to enter a nonalphanumeric character in a field that the data control type defines as an alphanumeric field.

Field overflow

The field is completely filled in; no more characters can be inserted.

Illegal function key

You pressed a key that the TabKey table considers to be illegal.

Invalid cursor movement

You tried to move the cursor:

- □ out of the field
- □ to the right of a decimal mark in a decimal field without typing a decimal mark (even if the decimal mark is already present)
- □ into an uninitialized part of a field

Make sure to move the cursor only within the allowable areas in a field.

Invalid data

The user validation routine rejected the current value of the field, but did not supply an error message.

Mandatory field

You have not filled in a mandatory field; please do so.

No list of values for this field

You pressed a Next Value or Previous Value key, while filling in a field that was not defined with a list of values.

No such value

For a field defined with a list of values, you typed an ASCII character that was different from the first character of any values in the list.

Numeric characters only

You tried to enter a nonnumeric character in a field that the data control type defines as a numeric field.

Reserved character

You tried to enter a character with ASCII codes 00h, 01h, or 03h into a field; you cannot use these characters as data characters in a field.

Sign cannot be inside a number

In a numeric field, you can enter a sign only as the first character.

Text characters only

You tried to enter a character into a field defined with the text (Tx) data control type, when you can enter a character only with the literal text entry method (CODE-=).

Sample BASIC Program

This BASIC program uses OpenForm, DisplayForm, UserFill Field, ReadField, WriteField, and UndisplayForm to run the form created in appendix D.

The program includes range checking and data validation of encoded data with the PartNumber field (lines 720-740).

Sample BASIC Program

This BASIC program uses OpenForm, DisplayForm, UserFill Field, ReadField, WriteField, and UndisplayForm to run the form created in appendix D.

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
20       OPTION BASE 1
30       CHNEXT% = &HA: ERCINVALIDDATA% = 3708
40       'Create part number <=> price data base
50       DIM PRICES% [20]
60       FOR I% = 1 TO 20:  READ PRICES%[I%]:  NEXT I%
70       DATA 1, 5, 17, 20, 3, 4, 7, 6, 2, 3, 3, 12, 15,
         19, 6, 8, 4, 9, 9, 15
80       '
100      'Open the form
110      FILE$ = "Tutorial.form"
120      PSWD$ = " "
130      FH% = 0
140      MODE% = &H6D72 ' modeRead
150      ERC% = OPENFILE(PTR(FH%), PTR(FILE$), LEN(FILE$),
         PTR(PSWD$), LEN(PSWD$), MODE%)
160      IF ERC% <> 0 THEN GOTO 9000
170      FORM$ = "Tutorial"
180      DIM FORM% [500] ' 500 words = 1000 bytes
190      ERC% = OPENFORM(FH%, PTR(FORM$), LEN(FORM$),
         PTR(FORM%[1]), 1000)
200      ERC2% = CLOSEFILE(FH%) ' close the file even if
         OpenForm got an error
210      IF ERC% <> 0 THEN GOTO 9000
220      IF ERC2% <> 0 THEN GOTO 9000
230      'Now clear the screen and display the form
         centered
240      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, PRT(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$(255) = "vn";
1515       ERC% = UNDISPLAYFORM(PTR(FORM%[1]))
1520       END
1530       '
5000       'Subroutine to display MSG$ 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
```

Sample PASCAL Program

```
PROGRAM TestPas (Input, Output);

 (* Program to exercise some of the procedures                  *)
 (* TypeForm, StoreFieldData and SetfieldType                   *)
 (* This program assumes the presence of the Form               *)
 (* 'Tutorial.form' constructed by following the instructions *)
 (*  for creating a form in appendix D.                         *)

  {$Debug-} (* Turn off the debugger *)

  TYPE
  1       Pointer                    = ADS of WORD;
          Quad                       = ADS of WORD;
          ErcType                    = WORD;
          FlagType= BOOLEAN;

VAR [PUBLIC]
          FileHandle: WORD             ;
          Index                      : WORD   ;
          Price                      : REAL   ;
          TotalPrice: REAL             ;
          AmountDue: REAL              ;
          UnitPrice: REAL              ;
          DeCodeReal: REAL             ;
          Part                       : INTEGER  ;
          Quantity: INTEGER           ;
          FrameNumber: INTEGER        ;
          DeCodeIntr: INTEGER         ;
          vtype                      : LSTRING(10)   ;
          Field                      : LSTRING(11)   ;
          NumStr                     : LSTRING(20)   ;
          StrCodeNum: LSTRING(20)     ;
          Prices                     : ARRAY[1..10] of REAL ;
          Form                       : ARRAY[1..2000] of BYTE;
          InitState: ARRAY[1..4] of WORD;
          ExitState: ARRAY[1..4] of WORD;
          FieldInfo: ARRAY[1..17] of WORD;
          RetBuf                     : ARRAY[1..4000] of BYTE;

VAR [EXTERNAL]
          BsVid                      : BYTE   ;


VAR
          i                          : WORD   ;

CONST
          Protected=                 2;
          Mandatory=                 1;
          ModeRead=                #6D72  ;
          InvalidData=             #3708  ;
          rgbCls                   = CHR(#0ff) * 'pf' * CHR(#0ff)
* 'vf' *
                                    CHR(#0ff) * 'c' * CHR(#0) *
CHR(#0) *
                                    CHR(#0ff) * 'ef' * CHR(#0);

(* Declare BTOS External Procedures *)

FUNCTION OpenFile(pFhRet           : Pointer ;
                                   pbFileSpec : Pointer ;
                                   cbFileSpec : WORD  ;
                                   pbPassWord : Pointer ;
                                   cbPassWord : WORD  ;
                                   Mode  : WORD  ) : ErcType;
EXTERN;

FUNCTION WriteBsRecord(pBSWA       : Pointer ;
                                   pb  : Pointer ;
                                   cb  : WORD  ;
                                   pcbRet  : Pointer ) : ErcType;
EXTERN;

FUNCTION CloseFile(FileHandle: WORD ) : ErcType; EXTERN;

PROCEDURE CheckErc(Erc             : WORD  ) ; EXTERN;

PROCEDURE Exit                        ; EXTERN;

(* Declare FORMS External Procedures *)

FUNCTION OpenForm(FileHandle: WORD  ;
                                   pbFormName : Pointer ;
                                   cbFormName : WORD  ;
                                   pFormRet : Pointer ;
                                   cbMax  : WORD  ) : ErcType;
EXTERN;

FUNCTION DisplayForm(pForm         : Pointer ;
                                   iFrame  : WORD  ;
                                   iCol  : WORD  ;
                                   iLine  : WORD  ) : ErcType;


EXTERN;

FUNCTION TypeForm(pForm            : Pointer ;
                                   pbRet  : Pointer ;
                                   cbMax  : WORD  ;
                                   pbFirstField : Pointer ;
                                   cbFirstField : WORD  ;
                                   index  : WORD  ;
                                   pbExitState : POINTER ) :
ErcType; EXTERN;

FUNCTION StoreFieldData(pForm      : Pointer ;
                                   pbFieldName : Pointer ;
                                   cbFieldName : WORD  ;
                                   index  : WORD  ;
                                   pbBuf  : Pointer ;
                                   pbRet  : Pointer ;
                                   cbMax  : WORD  ;
                                   pInfoRet : POINTER ;
                                   pType  : POINTER ) : ErcType;
```

```
EXTERN;

FUNCTION WriteField(pForm           : Pointer ;
                                    pbFieldName : Pointer ;
                                    cbFieldName : WORD  ;
                                    index  : WORD  ;
                                    pb   : Pointer ;
                                    cb   : WORD  ;
                                    pType  : POINTER ) : ErcType;
EXTERN;

FUNCTION SetFieldType(pForm         : Pointer ;
                                    fAll  : FlagType ;
                                    pbFieldName : Pointer ;
                                    cbFieldName : WORD  ;
                                    index  : WORD  ;
                                    fieldtype : WORD  ) : ErcType;
EXTERN;

(**********************    Main Program
*****************************)


BEGIN

(* Define pricing data *)
Prices[1] :=    9.95;
Prices[2] :=   15.87;
Prices[3] :=    3.31;
Prices[4] :=   19.78;
Prices[5] :=   12.23;

(* Clear the screen *)
CheckErc ( WriteBsRecord( Ads BsVid,
                                    Ads rgbCls,
                                    SizeOf (rgbCls),
                                    Ads i ));

(* Open the file containing the form *)
CheckErc ( OpenFile (Ads FileHandle,
                                    Ads 'Tutorial.Form',
                                    13,
                                    Ads ' ',
                                    0,
                                    ModeRead ) );

(* Read the form into local storage *)
CheckErc ( OpenForm (FileHandle,
                                    Ads 'Tutorial',
                                    8,
                                    Ads Form,
                                    SizeOf (Form) ) );

(* Free up the file *)
CheckErc ( CloseFile (FileHandle));

(* Throw the form up on the screen *)
CheckErc ( DisplayForm(Ads Form,
                                    0,
                                    255,
                                    255) );
```

```
(* Loop through the 5 rows of the form *)
FOR i := 1 TO 5 DO
          BEGIN

          Field:= 'UnitPrice';
          Vtype := 'Character.';

          (* Display the pricing data *)
          IF ENCODE (StrCodeNum, Prices[ORD(i)]:7:2)
          THEN BEGIN
               CheckErc ( WriteField (Ads Form,
                                    Ads Field[1],
                                    Field.len,
                                    i,
                                    Ads StrCodeNum[1],
                                    7,
                                    Ads Vtype[1]));
               END;

          (* Make the Unit Price field write protected *)
          CheckErc ( SetFieldType ( Ads Form,
                                    FALSE,
                                    Ads Field[1],
                                    Field.len,
                                    i,
                                    Protected));

          (* Make the Total Price field write protected *)
          Field:= 'TotalPrice';
          CheckErc ( SetFieldType ( Ads Form,
                                    FALSE,
                                    Ads Field[1],
                                    Field.len,
                                    i,
                                    Protected));
          END;

(* Make the Amount Due field write protected *)
Field     := 'AmountDue';
CheckErc ( SetFieldType ( Ads Form,
                                   FALSE,
                                   Ads Field[1],
                                   Field.len,
                                   0,
                                   Protected));
```

```
(* Make the Salesman field write protected *)
Field    := 'Salesman';
CheckErc ( SetFieldType ( Ads Form,
                                   FALSE,
                                   Ads Field[1],
                                   Field.len,
                                   0,
                                   Mandatory));

(* Now let Forms lead the user through the forms fill-in *)
CheckErc (TypeForm (Ads Form,
                                   Ads RetBuf,
                                   SizeOf (RetBuf),
                                   Ads Form,
                                   0,
                                   0,
                                   Ads ExitState) ) ;

(* The user has filled in the form -- now compute the Total
Prices *)
(* Note that data validation (either via TypeForm or
programmatic)
          is not implemented *)

Vtype     := 'Character.';
AmountDue := 0.0;

FOR i := 1 TO 5 DO
          BEGIN
          Field:= 'Quantity';

          (* Get the data entered for the Quantity field *)
          CheckErc (StoreFieldData ( Ads Form,
                                   Ads Field[1],
                                   Field.len,
                                   i,
                                   Ads RetBuf,
                                   Ads Numstr[1],
                                   7,
                                   Ads ExitState,
                                   Ads Vtype[1]) );
          Numstr.len := ExitState[1];
          (* Try to convert it to a numeric value *)
          IF DECODE (Numstr, Quantity)
          THEN BEGIN
               TotalPrice := Prices[ORD(i)] * Quantity;
               AmountDue := AmountDue + TotalPrice;

               (* Display the Total Price Field *)
               IF ENCODE (StrCodeNum, TotalPrice:7:2)
               THEN BEGIN
                                   Field := 'TotalPrice';
                                   CheckErc ( WriteField ( Ads
Form,
                                    Ads Field[1],
                                    Field.len,
                                    i,
                                    Ads StrCodeNum[1],
                                    7,
                                    Ads Vtype[1]) );
                                   END;
               END;

          END;

(* Now display the Amount Due value *)
IF ENCODE (StrCodeNum, AmountDue:7:2)
THEN BEGIN
          Field := 'AmountDue';
          CheckErc ( WriteField ( Ads Form,
               Ads Field[1],
               Field.len,
               0,
               Ads StrCodeNum[1],
               7,
               Ads Vtype[1]) );
          END;

(* Exit gracefully *)
Exit;

END.
```

Training Exercise: Creating a Form

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

If you are not familiar with the overtype and insert modes or cursor movement, you should review those paragraphs in section 3. Also refer to section 3 as necessary for additional information on the Forms Editor's functions.

Note: You can exit any of the Forms Editor's functions and return to the Forms Editor by pressing **CANCEL**. The **Test Drive** function is activated/deactivated using the (**F9**) function key. You can exit the Forms Editor and return to the Executive by pressing **FINISH**.

Activating the Forms Editor to Create a Form

To activate the Forms Editor, use the following procedure:

1 Enter **FORMS EDITOR** in the Executive command line.

2 Press **GO**.

The words **Forms Editor 6.0.0** appear at the top of the screen. The area below the line is work space for creating a form. The cursor is in the center of the screen.

Starting with Captions

To specify captions for the form, use the following procedure:

1 Move the cursor approximately 16 spaces to the left using either the arrow keys or the mouse.

2 Enter **Part No.** with a period (.) after the o.

3 Press **Right Arrow** three times.

4 Enter **Quan.** with a period (.) after the n.

5 Press **Right Arrow** three times.

6 Enter **Price**.

7 Press **Right Arrow** three times.

8 Enter **Total**.

The center of your form should now look like this:

```
Part No.   Quan.   Price   Total
```

Adding Lines and Boxes

To enclose the captions in a box, use the following procedure:

1 Position the cursor one line above and two character cells to the left of the **Part No.** caption.

2 Press **MARK**.

3 Position the cursor three lines below and two character cells to the right of the **Total** caption.

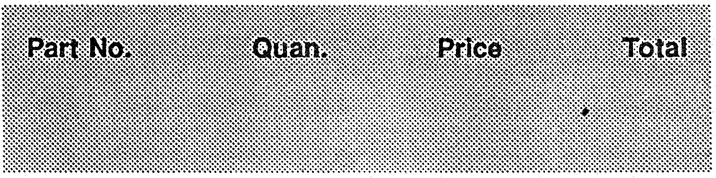
4 Press **BOUND**.

You have selected an area for a box. Your form should now look like figure D-1.

5 Press **SHIFT** and **DRAW** (**F4**).

The captions are surrounded by a thick line box. Your form should now look like figure D-2.

Figure D-1 **Selected Caption Box**



Part No. Quan. Price Total

To subdivide the box horizontally (that is, draw lines to separate the captions), use the following procedure:

1 Position the cursor one line below the captions at the right box boundary.

2 Press **MARK.**

3 Position the cursor on the same horizontal line (one line below the captions) at the left box boundary.

4 Press **BOUND.**

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

5 Press **Draw (F4).**

A thin horizontal line appears.

Figure D-2 **Captions in a Box**

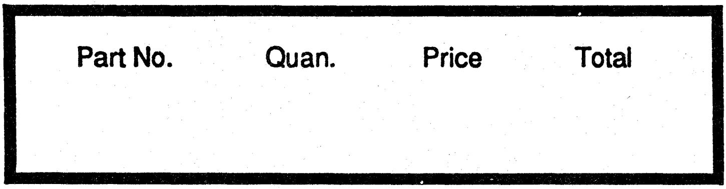
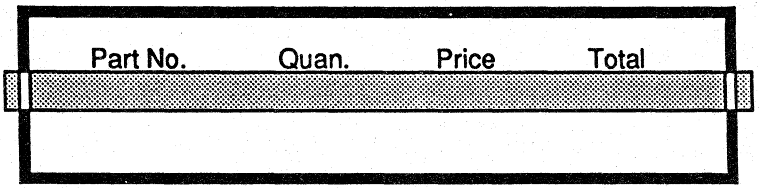


Figure D-3 **Selected Horizontal Line**



To subdivide the box vertically, use the following procedure:

1 Position the cursor midway between **Part No.** and **Quan.** and one line above the captions.

2 Press **MARK.**

3 Position the cursor in a direct line below the selected character cell, intersecting the bottom line.

4 Press **BOUND.**

5 Press **Draw (F4).**

A thin vertical line appears.

Use Reselect to add vertical lines between **Quan.** and **Price** and between **Price** and **Total** by doing steps 6, 7, and 8.

6 Press **Reselect (F1).**

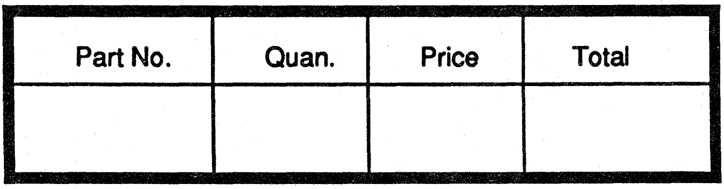
The selection for the line between **Part No.** and **Quan.** reappears.

7 Position the cursor one line above the captions, midway between **Quan.** and **Price.**

8 Press **COPY** twice. (You can use **COPY** because these columns are equal in area.)

Two more vertical lines appear. Your form should now look like figure D-4. The rightmost vertical line is still in the selected area.

Figure D-4 **Subdivided Box**



Part No.	Quan.	Price	Total

Adding Tabular Columns

To make a tabular form with five rows of data, you copy the lower box four times. **To add tabular columns, use the following procedure:**

1 To select the lower box using **MARK** and **BOUND**, position the cursor on the left boundary, one line below the captions. Press **MARK**.

2 Position the cursor on the lower right corner of the box. Press **BOUND**.

3 Position the cursor on the lower left corner of the box. (The cursor is just inside the lower left corner of the selection).

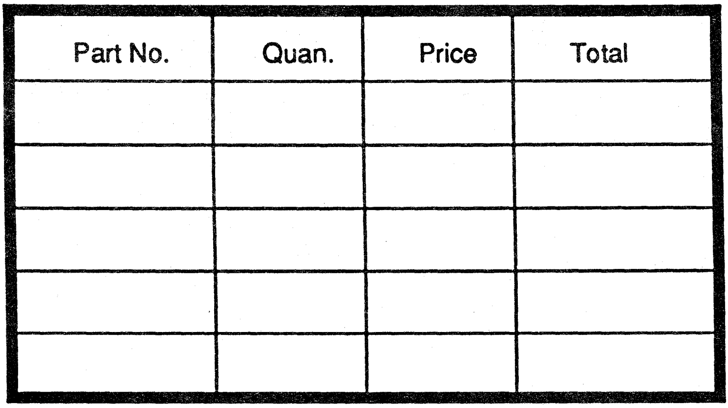
4 Press **COPY**.

A copy of the selected box is appended to the bottom of the box and is selected. The cursor is positioned on the lower left corner of the box (in the lower left corner of the selection).

5 Press **COPY** three more times.

Your form should now look like figure D-5. The bottom row of the form is selected.

Figure D-5 **Tabular Form**



Part No.	Quan.	Price	Total

Defining Fields for User Entry

Figure D-6 illustrates the define field command form and the bottom portion of your form with the **Part No.** repeating field selected.

To define repeating fields for each column, use the following procedure:

1 Select the column labeled **Part No.** as shown in figure D-6 by doing the following. Do not include the caption line in your selection.

Move the cursor to inside the left bottom corner of the form. Press **MARK**. The selected area disapppears.

2 Position the cursor two lines below the captions and to the left of the vertical line between the **Part No.** and **Quan.** columns. Press **BOUND**.

3 Press **Define Field (F8)**.

The Define Field command form appears. Your screen should now look like figure D-6.

4 Use **NEXT**, **RETURN**, or **Down Arrow** to move the cursor (left to right, top to bottom) through the fields of the Define Field command form. Use **Up Arrow** to move the cursor through the fields in reverse order.

When changing an existing entry (such as the **Yes** for **Show default?**), you must use **DELETE** to clear the character cells before you can enter a new entry.

5 Enter PartNumber in the **Name:** field of the Define Field command form.

6 Press **GO**.

Five fields are defined. Each contains a tag (sequence of square boxes).

7 Select the **Quan.** column. Press **Define Field (F8)** to define a repeating field named quantity with a default value of 1.

8 Press **GO**.

9 Select the **Price** column and define the repeating field **UnitPrice**.

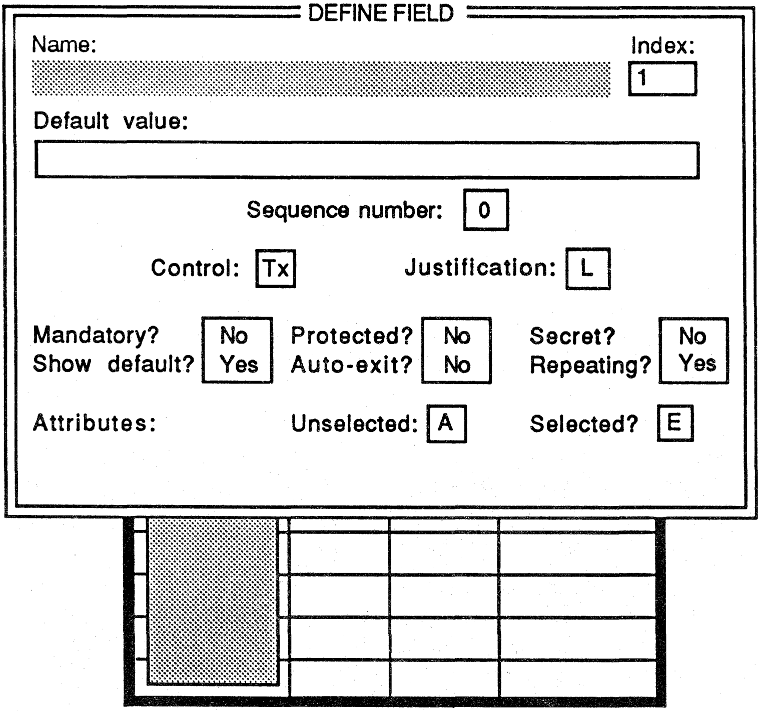
10 Press **GO**.

11 Select the **Total** column and define the repeating field **TotalPrice**.

12 Press **GO**.

Your form should now look like figure D-7.

Figure D-6 **Selection for a Repeating Field**



DEFINE FIELD

Name:

Index: 1

Default value:

Sequence number: 0

Control: Tx

Justification: L

Mandatory? No

Show default? Yes

Protected? No

Auto-exit? No

Secret? No

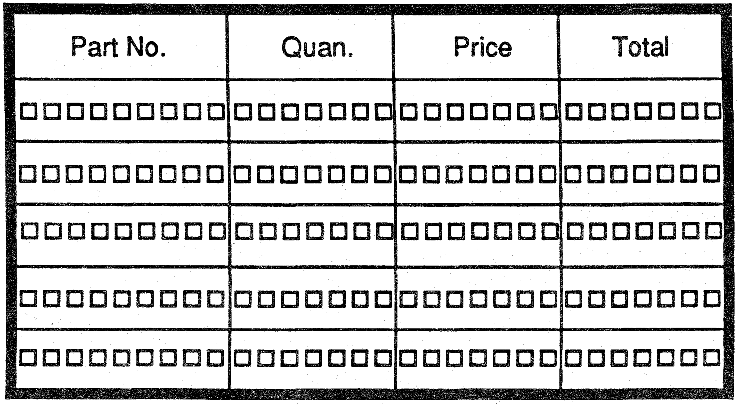
Repeating? Yes

Attributes:

Unselected: A

Selected? E

Figure D-7 **Defined Fields**



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

To define two single fields, use the following procedure:

1 Below the **Total** column, select an area three lines high (including the bottom line of the box and having the same width as the **Total** column).

2 Press **SHIFT-Draw (F4).**

A thick line box appears, joined to the main form.

3 Select the interior of this box and define a field named **AmountDue.**

4 Press **GO.**

5 Enter the caption **Amount Due:** to the left of the box.

6 Move the cursor to the top of the form.

7 Enter the caption **Salesman:** above the **Part No.** column.

8 Select a field one character high beginning to the right of the caption **Salesman:** and extending to just inside the right edge of your form.

9 Define a field named **Salesman** to the right of the caption.

10 Change the unselected character attribute to underline (letter C).

11 Press **GO.**

Your form should now look like figure D-8.

12 To save your form in a file named Tutorial by using the **Write Form** function, press **F7.**

13 Press **GO.**

14 To run a Test Drive on your form, press **F9.** Use **NEXT, RETURN,** or **Down Arrow** to move the cursor through the fields of your form, left to right, top to bottom. Use **Up Arrow** to move the cursor through the fields in reverse order.

15 Press **F9** to exit Test Drive.

16 Exit the Forms Editor by pressing **FINISH.**

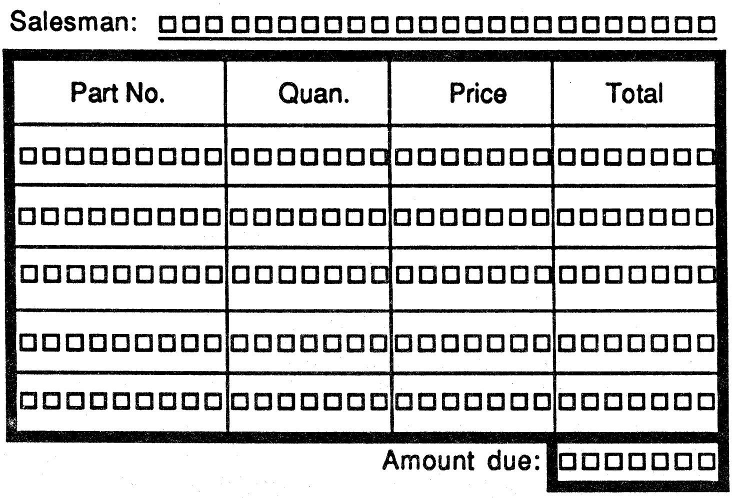
17 Activate the Forms Reporter by entering **FREPORT** in the Executive command line. Press **GO.** Enter the name of the saved file, Tutorial, in the required file parameter entry field. Press **GO.**

The report shown in figure D-9 is displayed.

Note: If the form is created using color palette information (**CODE-F8**), the color palette information for the form and the selected color numbers for lines and captions are shown at the end of the report.

18 Use the BASIC program in appendix B to run your form.

Figure D-8 **Finished Form**



Salesman: □□□ □□□□□□□□□□□□□□□□□□□□□

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

Amount due: □□□□□□□

Figure D-9 **Sample Form Report** (page 1 of 8)

```
Form name: tutorial                              size:   883 bytes
height:  16   width:  36   number of fields:    22

Field name: Salesman
Row:   0  Column:  13  Width:  22
Repeating? No   Index:          (first:         last:       )
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No  Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0        Unselected Color Number:  0

Field name: PartNumber
Row:   4  Column:   1  Width:  10
Repeating? Yes  Index:      1  (first:      1  last:      5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No  Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0        Unselected Color Number:  0

Field name: Quantity
Row:   4  Column:  12  Width:   7
Repeating? Yes  Index:      1  (first:      1  last:      5)
Control: Nu  Justification: R   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default: 1
Show default? Yes  Auto-exit? No  Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0        Unselected Color Number:  0
```

Figure D-9 **Sample Form Report** (page 2 of 8)

```
Field name: UnitPrice
Row:   4  Column:  20  Width:   7
Repeating? Yes  Index:     1  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: TotalPrice
Row:   4  Column:  28  Width:   7
Repeating? Yes  Index:     1  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: PartNumber
Row:   6  Column:   1  Width:  10
Repeating? Yes  Index:     2  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:            Unselected Color Number:  0
```

Figure D-9 **Sample Form Report** (page 3 of 8)

```
Field name: Quantity
Row:   6  Column:  12  Width:   7
Repeating? Yes  Index:     2  (first:     1  last:     5)
Control: Nu  Justification: R   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default: 1
Show default? No   Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: UnitPrice
Row:   6  Column:  20  Width:   7
Repeating? Yes  Index:     2  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: TotalPrice
Row:   6  Column:  28  Width:   7
Repeating? Yes  Index:     2  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0
```

Figure D-9 **Sample Form Report** (page 4 of 8)

```
Field name: PartNumber
Row:   8  Column:   1  Width:  10
Repeating? Yes  Index:     3  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: Quantity
Row:   8  Column:  12  Width:   7
Repeating? Yes  Index:     3  (first:     1  last:     5)
Control: Nu  Justification: R   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default: 1
Show default? No   Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: UnitPrice
Row:   8  Column:  20  Width:   7
Repeating? Yes  Index:     3  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0
```

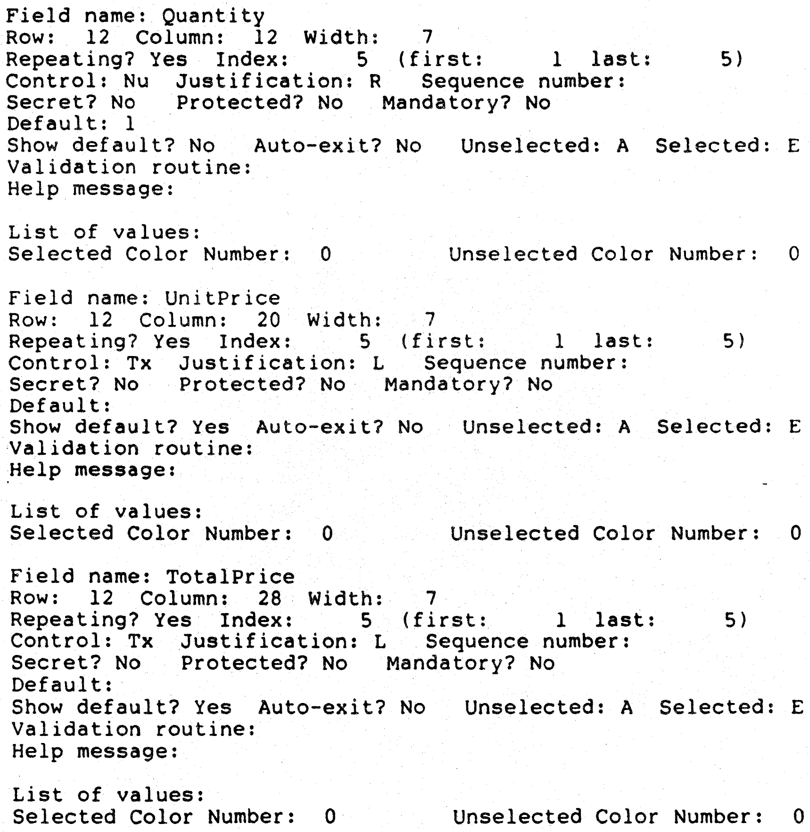
Figure D-9 **Sample Form Report** (page 5 of 8)

```
Field name: TotalPrice
Row:   8 Column:  28  Width:   7
Repeating? Yes  Index:     3  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: PartNumber
Row:  10  Column:   1  Width:  10
Repeating? Yes  Index:     4  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: Quantity
Row:  10  Column:  12  Width:   7
Repeating? Yes  Index:     4  (first:     1  last:     5)
Control: Nu  Justification: R   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default: 1
Show default? No   Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0
```

Figure D-9 **Sample Form Report** (page 6 of 8)

```
Field name: UnitPrice
Row:  10  Column:  20  Width:   7
Repeating? Yes  Index:     4  (first:     1  last:     5)
Control: Tx  Justification: L  Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: TotalPrice
Row:  10  Column:  28  Width:   7
Repeating? Yes  Index:     4  (first:     1  last:     5)
Control: Tx  Justification: L  Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: PartNumber
Row:  12  Column:   1  Width:  10
Repeating? Yes  Index:     5  (first:     1  last:     5)
Control: Tx  Justification: L  Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0
```

Figure D-9 **Sample Form Report** (page 7 of 8)



```
Field name: Quantity
Row:  12  Column:  12  Width:   7
Repeating? Yes  Index:     5  (first:     1  last:     5)
Control: Nu  Justification: R   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default: 1
Show default? No   Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: UnitPrice
Row:  12  Column:  20  Width:   7
Repeating? Yes  Index:     5  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0

Field name: TotalPrice
Row:  12  Column:  28  Width:   7
Repeating? Yes  Index:     5  (first:     1  last:     5)
Control: Tx  Justification: L   Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0
```

Figure D-9 **Sample Form Report** (page 8 of 8)

```
Field name: AmountDue
Row:  14  Column:  28  Width:   7
Repeating? No   Index:        (first:        last:       )
Control: Tx  Justification: L  Sequence number:
Secret? No   Protected? No   Mandatory? No
Default:
Show default? Yes  Auto-exit? No   Unselected: A  Selected: E
Validation routine:
Help message:

List of values:
Selected Color Number:  0         Unselected Color Number:  0
```

Glossary

AddValues. AddValues is a Forms Run-Time service that adds or modifies values which were created when an Editor **F8**–function-key default action defined a field.

Character Attribute. A character attribute is a visual highlight which can be applied to form fields and selected text or captions. Attributes include blinking, reverse video, half–bright, underlining, and combinations of these.

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 is a Forms Editor function that sets field characteristics such as field name, default, character attributes, auto exit, and repeating field.

Delete Selection. Delete Selection is a Forms Editor function that removes all captions, lines, and fields from a selected area.

DisplayForm. DisplayForm is a Forms Run-Time service that displays a form at a specific location.

Draw. Draw is a Forms Editor function that allows you to draw a line or box at the selected area.

EditForm. EditForm is a Forms Run-Time service that displays a form on the video or writes it to a file or device.

Erase. Erase is a Forms Editor function that allows deletion of a line or box from the selected area of a form but leaves the text and/or captions in place.

ExtractValues. ExtractValues is a Forms Run-Time service that returns a buffer containing the default values for each field of a form.

Field. A field is the specific area of a form that is set aside for accepting user data entries or displaying computed data. A form may contain many nonoverlapping fields, limited by display size and system byte availability. See DefaultField, Define Field, GetFieldInfo, ReadField, Repeating Field, SetFieldAttrs, UserFillField, WriteField.

FonctForm. FonctForm is a Forms Run-Time service that allows the application program to dynamically redefine some of the function keys.

Form. A form is a lined and captioned display with fields for accepting user data entries or displaying computed data. See DefaultForm, DisplayForm, Forms Reporter, Forms Run Time, OpenForm, Read Form, TypeForm, UnDisplayForm, Write Form.

Forms Reporter. Forms Reporter is a display that provides the following information for an existing form: 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.

Graphics Display. Graphics Display is a Forms Editor function that displays a graphics picture within a form.

Index Number. An Index Number is a number used to distinguish locations of repeating fields.

Insert Mode. Insert Mode is an edit mode that allows the insertion of characters at the current cursor position.

LockKbd. LockKbd is a Forms Run-Time service that interrupts keyboard entries until the user presses **CANCEL**.

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

OutForm. OutForm is a Forms Run-Time service that reads information about a form.

Overtype Mode. Overtype Mode is an edit mode that allows the replacement of characters at the cursor position.

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

Read Form. Read Form is a Forms Editor function that displays a stored form.

Repeating Field. A repeating field is a field that has the same name as another field. These fields are distinguished by index (location).

Reselect. Reselect is a Forms Editor function that allows repeated selection of the same form area.

RestoreForm. RestoreForm is a Forms Run-Time service that restores a form to its default values or to values supplied by the application program.

Selection. A selection is a rectangular area of the display that may be chosen by using **MARK** and **BOUND** or the mouse.

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

SetFieldType. SetFieldType is a Forms Run-Time service that allows the application program to dynamically modify the type of a field.

StoreFieldData. StoreFieldData is a Forms Run-Time service that extracts the value of a given field from a buffer returned by TypeForm.

Test Drive. Test Drive is a Forms Editor function that allows the user to display the form as it appears at run time and to fill in fields.

Text Insertion. Text Insertion is a Forms Editor function that recalls a previously saved text file after the Forms Editor has been activated.

TypeForm. TypeForm is a Forms Run-Time service that allows program input to form fields.

UnDisplayForm. UnDisplayForm is a Forms Run-Time service that removes a form from the display.

Undo. Undo is a Forms Editor function that deletes the previous change to a form.

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

Write Form. Write Form is a Forms Editor function that saves a form.

WriteField. WriteField is a Forms Run-Time service that writes from program memory.

Index

Errata Sheet

Errata Sheet for document:

BTOS Forms Designer Programming Guide
Relative to release level 6.0
Form 5027212
October 1987

Please add the following information to your copy of the guide described above.

******************************** page 3-2 ********************************

Add the following sentence to the first paragraphs of steps 6 and 7:

This step applies only to Forms Editor with graphics capability.

******************************** page 3-3 ********************************

Delete the following parameters from figure 3-2, FORMS EDITOR Command Form without Graphics:

[Column]
[Line]

****

UNISYS

Unisys Corporation
Software & Publications Distribution Services
41100 Plymouth Road, Building 4
Plymouth, Michigan 48170

BTOS Forms Designer 6.0.1

Enclosed is the 6.0 release of the BTOS Forms Designer software. We hope this program product will become an integral part of your business.

This product package consists of:

- Two 5-1/4-inch diskettes containing the 6.0 release of the BTOS Forms Designer
- BTOS Forms Designer 6.0 Release Notes (form 109200913)
- *BTOS Forms Designer Programming Guide*, release 6.0 (form 5027212)

You can order additional, priced copies from:

Unisys
Publication Center
41100 Plymouth Road
Plymouth, MI 48170

using order form #3020003, obtained from your local Unisys representative.

BTOS Forms Designer Release Notes

Style: SB20FM6, MB20FM6, XB20FM6

Level: 6.0.1

October 21, 1987

CONTENTS

BTOS Forms Editor and the Forms Run-Time for release 6.0 support additional hardware for design and development of forms. BTOS Forms Designer uses color on color-capable workstations and allows you to interactively select the new color assignments and view the result. There are new function keys to enhance the flexibility of designing forms.

NEW FEATURES

The new features in this release are:

- BTOS Forms Editor and Forms Run-Time support the B25 MD4 and the B25 GRC bit-mapped graphics hardware.

- New function keys are defined to allow more functionality in the Forms Designer:

  - The HELP function key provides an on-line HELP screen.

  - The CODE-F8 function key allows you to select and modify the color palette for the entire form and specify the color for lines and captions.

  - The F3 function key displays a graphics picture if the Forms Designer with graphics functionality is installed.

  - SHIFT-F3 removes the display graphics picture.

  - The F8 Define Field function key allows you to assign the Selected/Unselected color number to a field.

- The Forms Reporter lists the palette information and selection for the entire form and the individual field color attributes.

- Mouse support is provided for cursor positioning if the Mouse Server is installed to handle cursor control and tracking.

- Forms Designer 6.0 offers two files for defining function keys:

  FmTabKey.Asm.....Forms version 4.0 functionality

  FmTabKey5.Asm....Forms version 5.0 functionality

  The Forms.Lib provided in the 6.0 release has the 4.0 function key functionality. You can access Forms 5.0 function key functionality by assembling the FmTabKey5.Asm and adding it to the Forms.Lib as FmTabKey.Obj.

PRODUCT IMPROVEMENTS

The following are the major product improvements in this release:

- Delete Character key functionality has been added.
- The procedural interface for OutForm has been corrected.
- The error message display when using the herald area has been corrected.
- Validation for the index and sequence number fields of the define field (F8) function has been added.
- Cursor movement for Shifted Up and Down Arrows and the Left and Right Arrows with standard values has been corrected.
- The previous problem with using the Read Form (F6) function to detect invalid form names has been corrected.

KNOWN LIMITATIONS

- Forms Designer does not support Graphics on B25 bit-mapped graphics or B27 systems. Therefore, you should install Forms Designer without the Graphics capability option on these systems.

- When BTOS Forms Editor discovers an error in the specification of a form (e.g., duplicate field names), the unselected attribute of the offending field is changed to "I." This causes it to blink for easy identification. This attribute should be changed back to the desired setting when Define Field (**F8**) is used to correct the error. You can also use (**F2**) to remove the effects of the most recent function and to undo the blinking attribute.

- You can obtain vertical lines by using the character 16h, which is identical to vertical line Elh. You get the character 16h by typing CODE = and pressing function key **F2** where you want a thin vertical line.

- Because of hardware differences, you may not be able to create or edit forms on one workstation and use them on another.

  If you expect your forms to be used across the equipment line, then you should create your forms to the limits of the smallest screen. For example, if you are using B21, B22, B26, and B27 hardware, the form should be a maximum of 80 x 28 for the B21 screen (refer to the table on the following page).

  Also, you can use forms created in Zoom mode only on workstations that support Zoom mode (B22 and B27).

BTOS Workstation	Maximum Screen Size	Forms created to use maximum screen size can be used on:
B21	80 x 28	B21, B22, B26, B27, B28, or B38
B22	80 x 34 132 x 34	B22 or B27
B27	80 x 30 80 X 34 132 x 30 132 x 34	B22 or B27
B25 GRC	146 x 38	B25 GRC
B26, B28 or B38	80 x 29	B22, B26, B27, B28, or B38

In summary, if forms are to be used on different hardware types, you should use the smallest screen size in the group.

KNOWN PROBLEMS

These problems are listed in the Product Support Information Manual (PSIM); fix schedules will be maintained in the PSIMs.

Splitting a Field

Splitting a field into two parts usually causes the first part of the tag to become the field and the second part to become merely a caption. Duplicate field definitions sometimes result in an error message.

You should move a selection from one line to another if the selection contains a partial field. The Forms Editor sometimes allows move operations. The moved tag then becomes a caption that is disassociated from any field definition.

Test Drive

You should use Test Drive immediately following the error message **Too many fields**. This error message can result if you have many field definitions, usually caused by extensive field moving. Use Test Drive promptly to eliminate these definitions.

SOFTWARE FILES

The software is packaged on two 5-1/4-inch diskettes, as follows:

[B20FM6-1]<UNISYS>

FormsReporter.Run
FormsEditor.Run
sFormsEditor.Run

[B20FM6-2]<UNISYS>

Forms.Lib
FmrGtd.Asm
FmTabKey.Asm
FmTabKey5.Asm
FmValProc.Asm