The FormsVBT Reference Manual

Version 2.2

Marc H. Brown and James R. Meehan

March 26, 1993

©Digital Equipment Corporation 1992-1993

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of the Systems Research Center of Digital Equipment Corporation in Palo Alto, California; an acknowledgment of the authors and individual contributors to the work; and all applicable portions of the copyright notice. Copying, reproducing, or republishing for any other purpose shall require a license with payment of fee to the Systems Research Center. All rights reserved.

A. An Annotated Example

In this appendix we present a complete example of a non-trivial form, the one used for FormsEdit itself. This form uses nearly all of the FormsVBT components, as well as macros.

It will be easier to understand this form if you can run the FormsEdit program at the same time, to see what each part of the description looks like in the actual application.

The design of a user interface is not easy. We make no claims about this particular one; it has been used by a fair number people for over a year, although details have changed. You may have quite different preferences in fonts, colors, and layout. At the very least, FormsEdit will make it easy for you to "fix" this form!

In this appendix, we present the form hierarchically. We show "line numbers" along the left edge, and each line containing an ellipsis is explained in more detail in a subsequent section, where the line numbers use decimal points.

The following diagram shows the overall structure of the form, which is fairly typical: a top-level filter of some sort with the global properties; some macros; and a ZSplit with a background and dozen or so subwindows. While the entire file is fairly long, the structure is simple.

```
 1 (Shape %top ...
 2   (Macro TLA ...)
 3   (Macro TRA ...)
 4   (Macro SEP ...)
 5   (Macro BOX ...)
 6   (Macro COMMAND ...)
 7   (Macro FINDER ...)
 8   (Macro YESNO ...)
 9   (Macro CONFIRM ...)
10   (Macro FILEDIALOG ...)
11   (ZSplit
12     (ZBackground
13      (VBox (Glue 3)
14            (HBox %menubar ...)
15            (Glue 3)
16            Ridge
17            (TextEdit %buffer)
```

```
18            (FINDER ...)))
19      (ZChassis %manpage ...)
20      (ZChild %notFound ...)
21      (ZChild %aboutFE ...)
22      (ZChassis %errorPopup ...)
23      (ZChassis %PPwidthNumeric ...)
24      (ZChassis %snapshotDialog ...)
25      (ZChassis %dumpTablePopup ...)
26      (FILEDIALOG %OpenDialog ...)
27      (FILEDIALOG %SaveAsDialog ...)
28      (CONFIRM %quitConfirmation ...)
29      (CONFIRM %switchConfirmation ...)
30      (CONFIRM %closeConfirmation ...)
31      (YESNO %overwriteConfirmation ...)
32      (YESNO %RevertDialog ...)))
```

A.1 The top-level filter

The outermost form is usually a filter where you can place global properties. In this case, we use a Shape filter so that the editing windows can start out with a similar size.

```
1.0   (Shape %top
1.1     (Width  425 - 200 + Inf)
1.2     (Height 300 - 200 + Inf)
1.3     (LabelFont (Family "new century schoolbook"))
1.4     (Font "fixed")
1.5     (BgColor "PaleYellow")
1.6     (LightShadow "VeryVeryLightBlue")
1.7     (DarkShadow "Blue")
        ...)
```

On lines 1.1 and 1.2, we establish an initial size for the editor window, 425 points wide and 300 points tall. It can shrink to 200x200, and it can grow arbitrarily large. On lines 1.3 and 1.4, we specify the initial fonts for labels (Text forms) and for all the editable-text areas (TextEdit and TypeIn forms). Since the user will want to align columns of text (i.e., pretty-print the form), it is appropriate to use a fixed-width font. We also include the default colors here.

A.2 Simple macros

The first macros are just shorthand: TLA for “Text LeftAlign” and so on.

```
2 (Macro TLA BOA (x) `(Text LeftAlign ,x))
3 (Macro TRA BOA (x) `(Text RightAlign ,x))
4 (Macro SEP () '(VBox (Glue 3) Ridge (Glue 3)))
```

TLA and TRA are simple, 1-argument macros that are used several times throughout this form. BOA stands for "By Order of Argument," which means that the arguments are passed by position, not by keyword. This allows us to write `(TLA "Open")`, for example, instead of `(TLA (x "Open"))`.

SEP, on line 4, is the simplest form of macro, effectively a constant, since it takes no arguments. We use it to separate groups of items within a Menu.

A.3 A recursive macro

The BOX macro produces a series of nested, double-bordered boxes.

```
5.0 (Macro BOX (pens child)
5.1   (IF (= pens '())
5.2       child
5.3       `(Border
5.4          (Pen ,(List.Nth pens 0))
5.5            (Rim (Pen ,(List.Nth pens 1))
5.6              (BOX (pens ,(List.NthTail pens 2))
5.7                   (child ,child))))))
```

BOX is a recursive macro. It generates expressions of the form

```
(Border (Pen ...)
  (Rim (Pen ...)
    (Border (Pen ...)
      (Rim (Pen ...)
        ...)))))}
```

The `pens`-argument is a list of pen-widths; it must have an even number of elements. The even-numbered widths are used for the Borders; the odd-numbered widths are used for the Rims. For example,

```
(BOX (pens (2 4)) (child "Hello!"))
```

expands into

```
(Border (Pen 2) (Rim (Pen 4) "Hello!"))
```

The expression

```
(BOX (pens (2 4 5 10)) (child "Hello!"))
```

expands into

```
(Border (Pen 2)
  (Rim (Pen 4)
    (Border (Pen 5)
      (Rim (Pen 10)
        "Hello!"))))
```

Line 5.1 tests whether there are any pen-widths left in the list. If not, the expansion is simply the "child," on line 5.2. If there are pen-widths in the list, then the first two are used in lines 5.4 and 5.5, and the rest are passed recursively on line 5.6.

A.4 A macro for menu-items

The COMMAND macro generates menu-items that have some text on the left (the name of the command, such as "Open"), some filler-space in the middle, and a TSplit containing all the possible keybindings on the right. A keybinding is the name of a keyboard-equivalent for the command, such as **M-o**. FormsEdit allows you to change text-editing models, by means of a menu described in Section C.11 on page 167. When you do so, it changes all the keybindings in all the menus. In the Emacs model, for example, the keybinding for "Open" is written **M-o**, but in the Ivy model, it's **oO**.

```
6.0 (Macro COMMAND BOA (name label k1 k2 k3 k4 (type (MButton)))
6.1   `(,@type
6.2      ,name
6.3      (HBox
6.4        (TLA ,label)
6.5        Fill
6.6        (TSplit
6.7          %,(SxSymbol.FromName
6.8              (Text.Cat "Model_"
6.9                 (SxSymbol.Name (List.Nth name 1))))
6.10         (TRA ,k1)
6.11         (TRA ,k2)
6.12         (TRA ,k3)
6.13         (TRA ,k4)))))
```

The macro generates MButtons by default, as we see at the end of line 6.0. The button contains an HBox (line 6.3) that has the left-aligned command-label on the left (line 6.4) and the keybinding on the right (lines 6.6–6.13).

The arguments `k1`–`k4` are the keybindings for the four models. We use the TSplit on line 6.6 to switch among them. The name of the TSplit is constructed by concatenating the string `"Model_"` to the name of the menu-item, e.g., `"Model_Open"`. When the user changes the text-editing model,

the application calls `FormsVBT.PutInteger(fv, "Model_Open", n)` to set the TSplit to nth child, where n is the index that corresponds to the model.

Arguments are passed to this macro by position (BOA), and calls are written as

```
(COMMAND %foo ...)
```

That's equivalent to writing `(COMMAND (Name foo) ...)`, so the first parameter, `name`, is bound to the list `(Name foo)`. To extract the symbol `foo` from that, we call `(List.Nth name 1)` on line 6.9. Passing that symbol to `SxSymbol.Name` produces the string `"foo"`, which we then concatenate to the string `"Model_Open"` on line 6.8.

A.5 A macro for a Finder-dialog

To provide a user interface for text-searching, we use a "Finder," a form that contains a type-in field for the search-string, buttons for finding the first, next, or previous occurrence of that string, and a button to hide the form. See Figure C.1. The application searches through the text (the buttons tell it where to start and in which direction to look). If the search succeeds, the matching text is highlighted.

We use this form both in the main editing window and in the help-window, which is why we make it a macro.

```
7.0 (Macro FINDER (first next prev typein show close)
7.1   `(TSplit
7.2      Flex
7.3      Circular
7.4      (LightShadow "White")
7.5      (BgColor "VeryPaleBlue")
7.6      (Shape (Height 0) "")
7.7      (VBox %,show
7.8        Ridge
7.9        (Glue 2)
7.10       (HBox
7.11         (Shape (Width + 0) "Find:")
7.12         (Shape (Width 0 + 3000)
7.13           (Frame Lowered
7.14             (TypeIn %,typein (BgColor "VeryVeryPaleBlue"))))
7.15         (Glue 5)
7.16         (Shape (Width 0 + 1000)
7.17           (Rim (Pen 1) (Button %,first "First")))
7.18         (Shape (Width 0 + 1000) (ShadowSize 2.5)
7.19           (Button %,next "Next"))
7.20         (Shape (Width 0 + 1000)
```

Figure A.1: *The Finder dialog*

```
7.21           (Rim (Pen 1) (Button %,prev "Prev.")))
7.22         (Glue 20)
7.23         (Rim (Pen 1) (PageButton %,close "C")))
7.24       (Glue 2))))
```

The FINDER macro expands into a TSplit with two children. The first child, on line 7.6, has no size at all; since the TSplit has the Flex property (line 7.2), each child gets the size it wants, so "displaying" the first child is displaying nothing.

The second child is a VBox, beginning on line 7.7 (see Figure C.1). Since the Finder-dialog appears at the bottom of the editing and Help windows, we separate it from the rest with a Ridge (a 3-D bar) and 2 points' worth of Glue (lines 7.8–7.9). Underneath them is an HBox that contains a type-in field and three buttons: First, Next, and Previous. The Next button is the default, so it has the same effect as typing Return in the type-in field. To indicate that it is the default, it has no Rim, as the First and Prev buttons do (lines 7.17, 7.21). Instead, it uses the same amount of space for its shadow-size (line 7.18): 1 point to match the Rims' Pen-size, plus 1.5 points, which is the default shadow-size.

The (Width 0 + 3000) property on the framed type-in component (line 7.12), and the (Width 0 + 1000) properties on the three buttons, have the effect of diving the horizontal space into six equal regions (3 + 1 + 1 + 1), the first three of which are given to the type-in. These proportions will persist even if the subwindow grows or shrinks.

The "close button," marked "C," is a PageButton (line 7.23). Since the TSplit in which it appears has the Flex property, clicking this button selects the first child, the one that has no size, so the effect is to make the entire TSplit disappear.

A.6 A macro for yes/no dialogs

A yes/no dialog is a simple form containing two buttons; see Figure C.6 for an example. There are two places in the form where a yes/no dialog is needed (lines 31 and 32); we use this macro to ensure that the dialogs look the same.

```
8.0 (Macro YESNO (Name msg yesName noName)
8.1   `(ZChild %,Name
8.2      (BgColor "VeryPaleGray")
8.3      (LightShadow "White")
8.4      (DarkShadow "VeryDarkGray")
8.5      (Shape (Width 300)
```

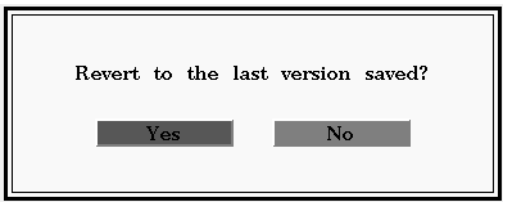

Figure A.2: *A yes/no dialog produced by the form*

```
(YESNO %RevertDialog
  (yesName revert)
  (noName dontRevert)
  (msg "Revert to the last version saved?"))
```

```
8.6         (BOX (pens (2 2 1 26))
8.7           (child
8.8             (VBox
8.9               ,msg
8.10              (Glue 20)
8.11              (HBox
8.12                Fill
8.13                (Button %,yesName (BgColor "Red")
8.14                  (Shape (Width 80) "Yes"))
8.15                Fill
8.16                (CloseButton %,noName (BgColor "Green")
8.17                  (Shape (Width 80) "No"))
8.18                Fill)))))))
```

The call to BOX on line 8.6 produces a 2-point Border around a 2-point Rim, which in turn encloses a 1-point Border around a 26-point Rim, which surrounds the VBox beginning on line 8.8.

The VBox contains a question (line 8.9), some filler, and two equally spaced buttons (8.13 and 8.16). We use the convention that the "safe" option is always green, and the "dangerous" option is red. In our case, the safe option is also the do-nothing option, so we use a CloseButton for "No" (line 8.16).

A.7 A macro for confirmation dialogs

A confirmation dialog is similar to a yes/no dialog, but it offers the user two choices on how to perform some action, plus a third choice of not performing the action at all. For example, the form

on line 30 asks the user whether changes should be saved before closing the file. The choices are: yes, close the file, but save the changes first; no, close the file, but discard the changes; and don't close the file at all. The third choice is often simply labeled "Cancel," but a more descriptive label (e.g., "Don't close") may be more helpful.

```
9.0 (Macro CONFIRM (Name question yesName noName
9.1                 cancelName cancelLabel)
9.2   `(ZChild %,Name
9.3      (BgColor "VeryPaleBlue")
9.4      (LightShadow "White")
9.5      (DarkShadow "VeryDarkBlue")
9.6      (Shape (Width 300)
9.7        (BOX (pens (2 2 1 26))
9.8          (child
9.9            (VBox
9.10             ,question
9.11             (Glue 20)
9.12             (HBox
9.13               Fill
9.14               (VBox
9.15                 (Button %,yesName (BgColor "Green")
9.16                   (Shape (Width 80) "Yes"))
9.17                 (Glue 10)
9.18                 (Button %,noName (BgColor "Red")
9.19                   (Shape (Width 80) "No")))
9.20               (Glue 20)
9.21               (VBox
9.22                 Fill
9.23                 (Filter %,cancelName
9.24                   (CloseButton (Shape (Width 80) ,cancelLabel))))
9.25               Fill)))))))
```

The arguments are the name for the subwindow, the question being asked (e.g., "Save changes before quitting?"), the names of the "yes," "no," and "cancel" button, and the text for the cancel button (e.g., "Don't quit").

Note that as in the YESNO macro on lines 8.0–8.18, green is used for the "safe" button, red for the "dangerous" button, but now the text of those buttons is reversed: the "safe" button says "No", and the "dangerous" button says "Yes", because the question being asked in a confirmation dialog is always "dangerous."

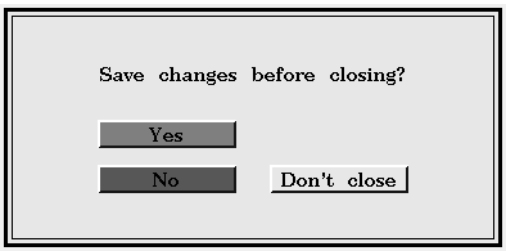

Figure A.3: *A confirmation dialog*

A.8 A macro for a file-chooser

As we mentioned in the description of the FileBrowser component on page 90, it is a good idea to combine a FileBrowser, a DirMenu, and a Helper, with activation and cancellation buttons, in a standard arrangement. We do that with the following macro.

```
10.0  (Macro FILEDIALOG
10.1    (Name BgColor DarkShadow Title fbName OKName OKLabel
10.2     cancelName (cancelLabel "Cancel") helperName
10.3     (ReadOnly #FALSE) (other ()))
10.4    `(ZChassis %,Name
10.5       (BgColor ,BgColor)
10.6       (LightShadow "White")
10.7       (DarkShadow ,DarkShadow)
10.8       (Title ,Title)
10.9       (Shape (Width 300 - 200 + Inf) (ShadowSize 2)
10.10        (Rim
10.11          (Pen 10)
10.12          (VBox
10.13            (HBox
10.14              (Shape (Width 150 + Inf) (Height 150 + Inf)
10.15                (VBox (LabelFont "fixed")
10.16                  (DirMenu (For ,fbName))
10.17                  (Glue 6)
10.18                  (Frame Lowered (BgColor "VeryPaleGray")
10.19                    (FileBrowser %,fbName))))
10.20              Fill
10.21              (Shape (Width 100)
10.22                (VBox
10.23                  Fill
```

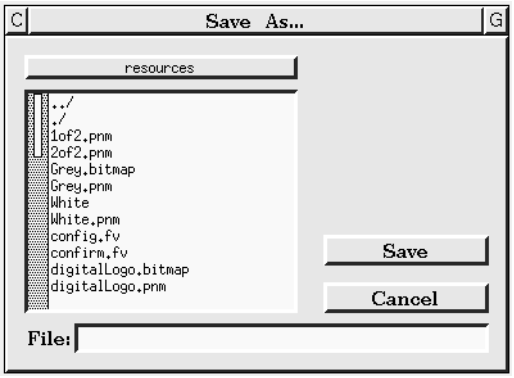

Figure A.4: *The 'Save As...' file-dialog*

```
10.24                  (Button %,OKName ,OKLabel)
10.25                  (Glue 10)
10.26                  (Filter
10.27                    (CloseButton %,cancelName ,cancelLabel)))))
10.28            (Glue 6)
10.29            (HBox
10.30              (Shape (Width 30) "File:")
10.31              (Frame Lowered
10.32                (Helper %,helperName FirstFocus (For ,fbName)
10.33                  (BgColor "VeryPaleGray"))))
10.34            ,@other)))))
```

While the overall appearance is standard, the names of the components, the colors, the labels, and other aspects may vary, so those are all passed in as parameters to the macro. Figure C.4 shows what this looks like for the "Save As..." dialog in Section C.24 on page 177.

The fixed-width label-font is used on line 10.15 to make it easier to read filenames. Filename-punctuation such as periods and slashes are often hard to read in small, variable-width fonts.

As in the example on line 9.22, the Filter on line 10.26 makes it possible to de-activate the Cancel button via `FormsVBT.MakeDormant`.

A.9 The background child

```
11   (ZSplit
12     (ZBackground
13       (VBox (Glue 3)
```

```
14              (HBox %menubar ...)
15              (Glue 3)
16              Ridge
17              (TextEdit %buffer)
18              (FINDER ...)))
        ...)
```

The background is a VBox with a menubar, the main text-editing area,[1] and the Finder-dialog.

A.10 The menubar

```
14.0  (HBox %menubar
14.1    (Glue 5)
14.2    (Menu ...)
14.3    (Glue 5)
14.4    (Menu ...)
14.5    (Glue 5)
14.6    (Menu ...)
14.7    (Glue 5)
14.8    (Menu ...)
14.9    (Glue 5)
14.10   Fill
14.11   (TSplit %modified
14.12     (Text "")
14.13     (Pixmap (Color "Red") "bnote.pbm"))
14.14   (Text %shortname "")
14.15   Fill
14.16   (Button %parse (BgColor "VeryPaleBlue")
14.17     (Text (Margin 10) "Do It"))
14.18   (Glue 5))
```

The menubar is an HBox with four Menus, a TSplit for the note-icon (14.11–14.13), the filename (14.14), and the "Do it" Button.

This TSplit displays either an empty string (14.12) or a small red warning note (14.13), depending on whether there are unsaved changes to the text in the buffer. The application switches between these choices by calling `FormsVBT.PutInteger(fv, "modified", n)`, where n is the index of the desired TSplit-child.

The name of the file is inserted into the menubar by assigning it to Text component on line 14.14, via `FormsVBT.PutText(fv, "shortname", ...)`. The Fill's on lines 14.10 and 14.15 cause the TSplit and the filename to be centered between the menus and the button.

[1] The most complex component has the shortest description in the form!

A.11 The quill-pen menu

```
14.2.0 (Menu
14.2.1   (Shape (Width 40) (Height 13.5) (Pixmap "pen.pbm"))
14.2.2     (Border
14.2.3       (VBox
14.2.4         (PopMButton (For aboutFE) (TLA "About FormsEdit..."))
14.2.5         (SEP)
14.2.6         (COMMAND %Help "Help..."
14.2.7           "oH" "M-h" "c-h" "M-h" (PopMButton (For manpage)))
14.2.8         Ridge
14.2.9         (Radio %Model
14.2.10          (HBox
14.2.11            (Glue 10)
14.2.12            (VBox
14.2.13              "Editing Model"
14.2.14              (Choice %ivyModel MenuStyle (TLA "Ivy"))
14.2.15              (Choice %emacsModel MenuStyle (TLA "Emacs"))
14.2.16              (Choice %macModel MenuStyle (TLA "Mac"))
14.2.17              (Choice %xtermModel MenuStyle (TLA "Xterm")))))
14.2.18        Ridge
14.2.19        (COMMAND %quit2 "Quit" "oQ" "M-q" "c-q" "M-q")))))
```

The quill-pen icon was created using `bitmap(1)`, converted to "pnm" format by `anytopnm(1)`, and saved in the resource-file named `pen.pbm`. All the menu-buttons have a width of 40 points.

Line 14.2.4 shows a simple pop-menu-button containing a left-aligned string. When the button is released, FormsVBT automatically opens the subwindow whose name in this form is `aboutFE`. The subwindow itself is defined on line 21.

The call to `(SEP)` on line 14.2.5 produces the VBox that separates groups within a menu. It was defined on line 4; see Section C.2 on page 158.

The first "command" button is the Help button on lines 14.2.6–14.2.7. As it happens, it is a PopMButton, not an MButton; by passing a seventh argument, `(PopMButton (For manpage))`, we avoid getting the default value for the `type` parameter; see line 6.0 in Section C.4 on page 159. The name of the button is `Help`. The four keynames are the keyboard shortcuts for the Ivy, Emacs, Mac,[2] and Xterm models, in that order.

This macro-call expands into the following expression:

```
(PopMButton (Name Help)
  (HBox (Text LeftAlign "Help...")
        Fill
```

[2] We use "c-h" for the Mac, since there is no ISO Latin-1 character that corresponds to the Mac "command" or "cloverleaf" icon.

```
        (TSplit (Name Model_Help)
           (Text RightAlign "oH")
           (Text RightAlign "M-h")
           (Text RightAlign "c-h")
           (Text RightAlign "M-h"))))
```

The 4 radio-buttons on lines 14.2.14–14.2.17 allow the user to choose the editing model. The group is indented slightly (line 14.2.11).

A.12 The File menu

The File menu has the standard assortment of items: Open, Save, Save As..., etc. The form describes all the visual aspects of the menu; the application interprets each of the commands by attaching an event-handler to each name.

```
14.4.0  (Menu
14.4.1    (Shape (Width 40) "File")
14.4.2     (Border
14.4.3       (Shape (Width 110)
14.4.4         (VBox
14.4.5           (COMMAND %new "New" "oN" "M-n" "c-n" "M-n")
14.4.6           (COMMAND %openMButton "Open..."
14.4.7             "oO" "M-o" "c-o" "M-o"
14.4.8             (PopMButton (For OpenDialog)))
14.4.9           (SEP)
14.4.10          (MButton %close (TLA "Close"))
14.4.11          (Filter (COMMAND %save "Save" "oS" "M-s" "c-s" "M-s"))
14.4.12          (PopMButton %saveasMButton (For SaveAsDialog)
14.4.13            (TLA "Save As..."))
14.4.14          (PopMButton %revertbutton (For RevertDialog)
14.4.15            (TLA "Revert To Saved"))
14.4.16          (SEP)
14.4.17          (PopMButton %ppwidthPopMButton (For PPwidthNumeric)
14.4.18            (TLA "PP setup..."))
14.4.19          (COMMAND %PPrint "PPrint" "oP" "M-p" "c-p" "M-p")
14.4.20          (SEP)
14.4.21          (COMMAND %quit "Quit" "oQ" "M-q" "c-q" "M-q"))))))
```

The Filter on line 14.4.11 enables the application to gray-out the Save button, which it does when no changes have been made to the buffer, sincethere's no point in saving an unmodified file.

The Quit button on line 14.4.21 is a duplicate of the item on line 14.2.19 at the bottom of the quill-pen menu. A Quit button normally appears at the bottom of the File menu, as it is here, but

some users expect to see it at the bottom of the top-left menu, whatever that may be called. The application attaches the same event-handler to both quit-buttons.

A.13 The Edit Menu

The Edit menu has a standard set of items, plus three additional items for searching.

```
14.6.1  (Menu
14.6.2    (Shape (Width 40) "Edit")
14.6.3      (Border
14.6.4        (Shape (Width 100)
14.6.5          (VBox
14.6.6            (COMMAND %undo "Undo" "cZ" "C-_" "c-z" "M-z")
14.6.7            (COMMAND %redo "Redo" "csZ" "M-_" "c-Z" "M-Z")
14.6.8            (SEP)
14.6.9            (COMMAND %cut "Cut" "oX" "C-w" "c-x" "M-x")
14.6.10           (COMMAND %copy "Copy" "oC" "M-w" "c-c" "M-c")
14.6.11           (COMMAND %paste "Paste" "oV" "C-y" "c-v" "M-v")
14.6.12           (COMMAND %clear "Clear" "" "" "" "")
14.6.13           (COMMAND %selectAll "SelectAll"
14.6.14             "oA" "M-a" "c-a" "M-a")
14.6.15           (SEP)
14.6.16           (COMMAND %findMButton "Find..."
14.6.17             "oF" "" "c-f" "M-f"
14.6.18             (LinkMButton (For FindInBuffer2)))
14.6.19           (COMMAND %findNext "Find Next" "c," "C-s" "" "")
14.6.20           (COMMAND %findPrev "Find Prev" "cM" "C-r" "" "")))))
```

The “Find” button on 14.6.18 is a not a PopMButton controlling a subwindow, although that would be more typical. Instead, it is a LinkMButton that selects the second child (see lines 7.7–7.23) of the Finder-dialog at the bottom of the window. The Finder is a TSplit; when its second child appears, the main window shrinks somewhat, but nothing is hidden. The typical pop-up window would *overlap* the main window and could easily obscure the text that the user is highlighted after a successful search.

A.14 The Misc Menu

```
14.8.0   (Menu
14.8.1     (Shape (Width 40) "Misc")
14.8.2       (Border
14.8.3         (VBox
```

```
14.8.4          (PopMButton %dumpTable (For dumpTablePopup)
14.8.5            "Show the named VBTs...")
14.8.6          (PopMButton %snapshot  (For snapshotDialog)
14.8.7            "Show current snapshot...")
14.8.8          (PopMButton (For errorPopup)
14.8.9            "Show last error message")
14.8.10         (SEP)
14.8.11         (Filter %rescreenFilter (VBox %rescreenMenu)))))
```

The "dumpTable" and "snapshot" buttons bring up windows that give information about the form; they are defined on lines 24–25.

Error messages are reported in a green subwindow in the bottom-right corner (see line 22), but the application automatically closes this window after 5 seconds. The button on line 14.8.8 gives the user a way to bring it back without a 5-second timeout, to read the message more carefully.

Note that the VBox on line 14.8.11 is empty. The application dynamically inserts MButtons here, one pair for each display-screen, for moving the editor window or the result window. On a 1-screen workstation, no buttons are inserted, and the VBox is grayed-out via the Filter. (Some window managers allow windows to be dragged between screens, which is more convenient than using these buttons.)

A.15 The Finder-dialog

The Finder at the bottom of the main window is generated by the following call:

```
18.0 (FINDER
18.1   (show FindInBuffer2)
18.2   (first bhelpfindfirst)
18.3   (next bhelpfindnext)
18.4   (prev bhelpfindprev)
18.5   (typein bhelpfindtext)
18.6   (close bhelpfindclose))
```

The layout is fixed (see Figure C.1); all the parameters are names assigned to the various components. The application attaches event-handlers to these names to perform the search.

A.16 The Help subwindow

```
19.0  (ZChassis %manpage
19.1    (BgColor "VeryPaleBlue")
19.2    (Title "formsedit help")
19.3    (VBox
```

```
19.4      (HBox
19.5        (Menu
19.6          (Shape (Width 40) "Edit")
19.7          (VBox
19.8            (COMMAND %mpcopy "Copy" "oC" "M-w" "c-c" "M-c")
19.9            (COMMAND %mpselectAll "SelectAll"
19.10             "oA" "M-a" "c-a" "M-a")
19.11           (SEP)
19.12           (COMMAND %mpfindMButton "Find..."
19.13             "oF" "" "c-f" "M-f"
19.14             (LinkMButton (For FindDialog)))
19.15           (COMMAND %mpfindNext "Find Next"
19.16             "c," "C-s" "" "")
19.17           (COMMAND %mpfindPrev "Find Prev"
19.18             "cM" "C-r" "" "")))
19.19       Fill)
19.20     Ridge
19.21     (Shape (Width 360 + Inf) (Height 150 + Inf)
19.22       (TextEdit ReadOnly %manpagetext))
19.23     (FINDER
19.24       (show FindDialog)
19.25       (first helpfindfirst)
19.26       (next helpfindnext)
19.27       (prev helpfindprev)
19.28       (typein helpfindtext)
19.29       (close helpfindclose))))
```

This is the subwindow that pops up when the user clicks the Help button on line 14.2.6. (See Figure C.5.) Its structure is similar to that of the main window. It contains a menubar (HBox) with one menu, Edit, which has items appropriate for a read-only buffer (e.g., Copy but not Cut). The Edit menu also includes a LinkMButton for another Finder-dialog, just as the main editing window does. Below the menubar is a read-only text-editor, which the application fills with the manpage for FormsEdit. At the bottom of the subwindow is the Finder-dialog, which is shown in the open position in Figure C.5.

A.17 A disappearing subwindow

```
20.0  (ZChild %notfound (BgColor "Red") (Color "White")
20.1    (Rim (Border "Not found")))
```

This is a small subwindow that the application displays—there is no PopButton for this component—whenever a search has failed, either in the editor window or in the “manpage” window. The

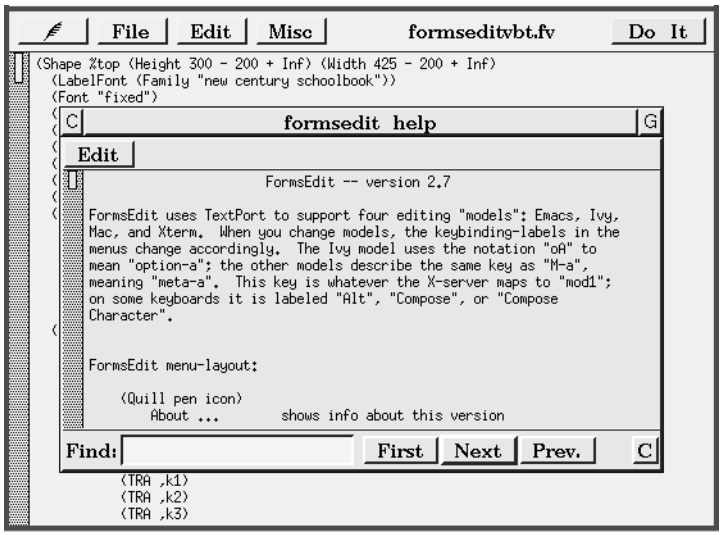

Figure A.5: *The "manpage" window*

application removes it after 2 seconds.

A.18 The About... window

```
21.0  (ZChild %aboutFE (BgColor 0.8 0.8 1)
21.1    (BOX (pens (1.5 2 1 12))
21.2      (child
21.3        (VBox
21.4          (Pixmap (Color "Blue") "digitalLogo.pbm")
21.5          (Glue 6)
21.6          "FormsEdit version 2.7"
21.7          "Written by Jim, Marc, and Steve."
21.8          "Copyright \251 1993 Digital Equipment Corp."
21.9          "Send comments to meehan@src.dec.com"
21.10         (Glue 6)
21.11         Ridge
21.12         Ridge
21.13         (Glue 6)
21.14         (HBox
21.15           Fill
```

Figure A.6: *The "About FormsEdit..." window*

```
21.16           (CloseButton (BgColor "VeryPaleBlue")
21.17                        (LightShadow "White")
21.18             (Text (Margin 5) "Close"))
21.19           Fill)))))
```

This is a completely static window. Note that the CloseButton (21.16) does not need a For- property, since it closes the subwindow that contains it. The use of the BOX macro (21.1) produces the double-bordered effect. (The 4-character sequence `\251` in the middle of the text on line 21.8 is converted into a single character by the S-expression reader; that character's code is 251_8, which is the ISO Latin-1 standard code for the copyright symbol, ©.)

A.19 The error-message subwindow

```
22.0  (ZChassis %errorPopup
22.1    (At 1. 1. SE)
22.2    (BgColor "VeryPaleGreen")
22.3    (Title "Error")
22.4    (LightShadow "White")
22.5    (DarkShadow "DarkGreen")
22.6    (Shape (Width 300 + Inf - 200) (Height 50 + Inf - 50)
22.7      (TextEdit %stderr ReadOnly)))
```

The error-message window is displayed by the application whenever there is a parsing error. The application also removes it after 5 seconds. (The Misc menu has an item to make this window re-appear.)

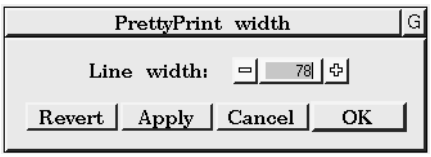

Figure A.7: *The "PrettyPrint width" window*

A.20 The pretty-print-width subwindow

```
23.0  (ZChassis %PPwidthNumeric
23.1    NoClose
23.2    (BgColor "PaleGold")
23.3    (Title (Text "PrettyPrint width"))
23.4    (At 0.1 0.1 NW)
23.5    (Shape (Width 250 + Inf)
23.6      (Rim
23.7        (Pen 10)
23.8        (VBox
23.9          (HBox
23.10           Fill
23.11           (Shape (Width 80) "Line width:")
23.12           (Glue 10)
23.13           (Shape (Width 70)
23.14             (Numeric %ppwidth FirstFocus
23.15               (BgColor "VeryPaleGold")
23.16               =78 (Min 30) (Max 200)))
23.17           Fill)
23.18         (Glue 10)
23.19         (HBox
23.20           (Shape (Width 0 + 1000)
23.21             (Rim (Pen 1) (Button %ppwRevert "Revert")))
23.22           (Shape (Width 0 + 1000)
23.23             (Rim (Pen 1) (Button %ppwApply "Apply")))
23.24           (Shape (Width 0 + 1000)
23.25             (Rim (Pen 1) (CloseButton "Cancel")))
23.26           (Shape (Width 0 + 1000) (ShadowSize 2.5)
23.27             (Button %ppwOK "OK")))))))
```

Some people would argue that having a pretty-printer built in to the editor is an essential tool in one's programming environment, regardless of the source language. It is especially helpful when

the entire program is a single expression. (A parenthesis-balancer would help, too!) This pop-up window allows the user to specify the maximum line-width that the pretty-printer will use, measured in characters. (The pretty-printer is available as the "PPrint" item in the file menu; see line 14.1.19.) It is sometimes helpful, especially on a 2-screen workstation, to make the window as wide as possible and then to increase the pretty-print width; 150 is a reasonable maximum.

This subwindow uses the NoClose property (see line 23.1), which removes the button labeled "C" from the top-left corner. We do this so that we can be more precise about the side-effects of closing the window. There are two ways to close it. Clicking the Cancel button closes it without permanently changing the desired width, that is, the width that will be used on all subsequent calls to the pretty-printer. Clicking OK, or typing Return in the Numeric type-in, will invoke the pretty-printer with the new width, setting that to be the desired width, and finally closing the subwindow.

On line 23.14, the FirstFocus property has the effect that the type-in field within the Numeric will grab the keyboard focus whenever this subwindow pops up, and it selects the text (the width) in replace-mode, to make it easy for the user to type a new width, hit Return, and have the form prettyprinted.

Note that the first three buttons have a 1-point Rim around them, but that the last button, OK, does not. Instead, it has a shadow that is 1 point larger than the default (which is 1.5 points). The effect is to make the OK button stand out a little more: it is the "default" button, so it has the same effect as typing Return in the Numeric. In 2-D style, as on a monochrome screen, "default" buttons are usually given a black border. This is the same convention we used in the Finder window; see Section C.5 on page 160.

A.21 The snapshot subwindow

```
24.0  (ZChassis %snapshotDialog
24.1    (At 0.1 0.9 0.2 0.8 Scaled)
24.2    (BgColor "VeryPaleTurquoise")
24.3    (Title (Text (BgColor "White") (Color "DarkTurquoise")
24.4      "Current Snapshot"))
24.5    (Shape (Height 250 - 100 + Inf)
24.6      (TextEdit %SnapshotText ReadOnly)))
```

This is a subwindow used primarily for debugging "snapshot" and "restore" operations. See Section 4.5 on page 66.

A.22 The named-components subwindow

```
25.0  (ZChassis %dumpTablePopup
25.1    (BgColor "PaleGold")
26.2    (At 0.1 0.9 0.2 0.8 Scaled)
26.3    (Title (Text (BgColor "White") (Color "Blue") "Named VBTs"))
```

```
26.4    (Shape (Height 300 - 100 + Inf)
26.5      (TextEdit %VBTtable ReadOnly)))
```

This is also primarily a debugging window, but it's useful for debugging layout problems. Each named component in the form is described on a separate line that includes its type, its size-range, and its actual size, in both the horizontal and vertical dimensions. For example, the following line appears for the ZChassis on line 23:

```
PPwidthNumeric : FVTypes.FVZChassis
        H: [330, 330, 100001] = 330. V: [113, 113, 114] = 113.
```

The name of the component is `PPwidthNumeric`. Its runtime type is `FVTypes.FVZChassis`. Its horizontal size-range has a "lo" and a "pref" of 330 pixels, with a very large "hi" value (essentially unlimited stretchability). Its actual width is 330. Its vertical size-range has no stretchability; "lo" and "pref" are 113 pixels, and "hi" is 114. Its actual size is 113. (The actual size will be 0 for components that are not visible.)

A.23 The open-file dialog

```
26.0  (FILEDIALOG %OpenDialog
26.1    (BgColor "VeryPaleGreen")
26.2    (DarkShadow "RatherDarkGreen")
26.3    (Title "Open an existing file")
26.4    (fbName openfile)
26.5    (ReadOnly #TRUE)
26.6    (OKName open)
26.7    (OKLabel "Open")
26.8    (cancelName cancelOpen)
26.9    (helperName fbh)
26.10   (other
26.11     ((Glue 6)
26.12      (HBox
26.13        (Radio =newwindow
26.14          (VBox
26.15            (Choice %reuse (TLA "Use this window"))
26.16            (Choice %newwindow (TLA "Open a new window"))))
26.17        Fill
26.18        (Radio =fvonly
26.19          (VBox
26.20            (Choice %fvonly (TLA "*.fv only"))
26.21            (Choice %notfvonly (TLA "Any file")))))))))
```

This is a standard file-dialog, produced by the macro on line 10, described in section C.8 on page 164.

As an addition to the standard file-dialog controls, this one allows the user to specify whether a new pair of windows (editor and result) should be used when displaying a new file. When this macro-call is expanded, the expressions on lines 26.11–26.21 are simply appended (see line 10.34) to the forms inside the VBox that starts on line 10.12.

The radio buttons on lines 26.15 and 26.16 are used by the application to determine whether to open the new file in a separate window.

The radio buttons on lines 26.20 and 26.21 are used to control the value that is passed to the procedure (`FileBrowserVBT.SetSuffixes`) that reads directories.

A.24 The save-as dialog

This window, shown in Figure C.4 on page 165, is simpler than the preceding open-file dialog: it has no "other" parameter, so it contains exactly what the macro specifies. It uses a different background color to distinguish it from the open-file dialog.

```
27.0  (FILEDIALOG %SaveAsDialog
27.1    (BgColor "VeryPaleBlue")
27.2    (DarkShadow "Blue")
27.3    (Title "Save As...")
27.4    (fbName saveasfile)
27.5    (OKName saveas)
27.6    (OKLabel "Save")
27.7    (cancelName cancelsaveas)
27.8    (helperName sfbh))
```

A.25 The confirmation dialogs

The three "confirmation" subwindows all perform a similar function, so the use of a macro makes it easy to give them a similar appearance. All that varies is the text they display. The macro itself is described in Section C.7 on page 163.

```
28.0  (CONFIRM %quitConfirmation
28.1    (question "Save changes before quitting?")
28.2    (yesName saveandquit)
28.3    (noName quitAnyway)
28.4    (cancelName dontquit)
28.5    (cancelLabel "Don't quit"))
29.0  (CONFIRM %switchConfirmation
29.1    (question "Save changes before switching?")
```

```
29.2    (yesName saveandswitch)
29.3    (noName switchAnyway)
29.4    (cancelName cancelSwitch)
29.5    (cancelLabel "Don't switch"))
30.0  (CONFIRM %closeConfirmation
30.1    (question "Save changes before closing?")
30.2    (yesName saveandclose)
30.3    (noName closeAnyway)
30.4    (cancelName cancelClose)
30.5    (cancelLabel "Don't close"))
```

A.26 The yes/no dialogs

The "yes/no" subwindows are similar to the "confirmation" windows. The macro is described in Section C.6 on page 162.

```
31.0  (YESNO %overwriteConfirmation
31.1    (msg "That file already exists. Overwrite it?")
31.2    (yesName overwrite)
31.3    (noName dontOverwrite))
32.0  (YESNO %RevertDialog
32.1    (yesName revert)
32.2    (noName dontRevert)
32.3    (msg "Revert to the last version saved?"))
```