VIB: A Visual Interface Builder for Icon

Mary Cameron and Gregg Townsend

Department of Computer Science, The University of Arizona

1. Introduction

Construction of an interactive graphics application can require a large effort to specify the screen layout. For graphics programs written in Icon [1][2], VIB eases this part of the project by providing an interactive layout tool.

An application built by VIB has two parts: the interface specification and the rest of the program. Both can be contained in the same source file. VIB edits the interface specification; any text editor can maintain the rest.

When VIB edits an existing file, it alters only the interface specification (adding one if necessary) and leaves the rest of the file unmodified. When VIB creates a new file, it writes a complete program by providing a skeletal Icon program to accompany the interface specification. This program simply prints each event (button push, slider movement, or whatever) on standard output, but it can be modified incrementally to produce the actions desired while remaining a functioning program at each stage.

VIB provides a prototyping facility for testing the interface at any time. The interface is built and executed to allow experimentation with the interface. Prototyping is possible regardless of the state of the Icon code in the file being edited.

Applications built by VIB use the visual interface (vidget) library [3]. Not all of the vidget capabilities are supported by VIB; sophisticated applications can call vidget procedures directly to augment the VIB-built interface.

In addition to laying out the application as a whole, VIB can also be used to construct dialog boxes. This process is described in Section 5.

VIB is a revision of an earlier interface builder, XIB [4]. VIB uses compact specifications that can coexist with user source code in the same file. The Icon program library contains a program uix for translating an XIB specification into VIB form.

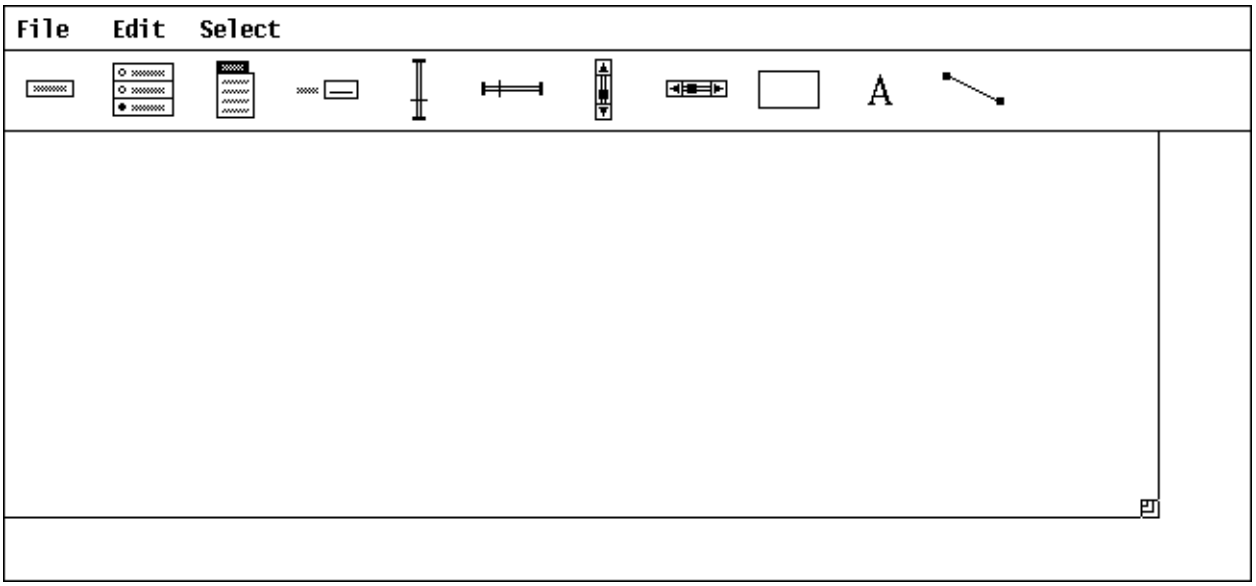
VIB is started by entering the command

vib [*filename*]

where *filename* is the name of an existing application source file or a new one to be created. If *filename* does not contain a suffix, .icn is appended. If *filename* is omitted, VIB generates a name.

2. The Palette

Upon startup, VIB presents a large window containing a menu bar, a palette of user interface objects, and a work area or canvas.



The palette contains pictorial icons for the following objects, from left to right:

Buttons are control devices that are activated by pressing the mouse while over the button. Several forms of buttons are available.

Radio Buttons are collections of buttons such that exactly one button within the radio button is set at any given time.

Menus are lists of buttons that appear temporarily on the screen and allow the user to select an item.

Text Input Fields gather textual input from the user. They consist of a label and an editable value.

Sliders select a scalar value within a range. Vertical and horizontal sliders are available.

Scrollbars also select values in a range, but provide buttons at both ends as well as a slider. Vertical and horizontal scrollbars are available.

Regions delimit rectangular areas of the window. The application can safely draw in these areas and receive uninterpreted events.

Labels are read-only text; they do not receive events.

Lines are provided to decorate the interface; they do not receive events.

Instances of the above objects can be created by pressing the left mouse button upon the desired palette object; this creates an object within the canvas area that then can be dragged into position. Clicking the left mouse button upon an object *selects* the object. Selected objects are drawn with accentuated corners and can be manipulated as follows:

move drag left mouse button from object to new destination.

nudge press an arrow key (*up, down, left* or *right*) to move the object by one pixel.

resize drag left mouse button upon a corner of the object. The opposite corner is anchored.

copy select *copy* from the **edit menu**.

delete select *delete* from the **edit menu**.

align select *align vert* or *align horz* from the **edit menu**. This causes the mouse cursor to change to a crosshair; pressing the left mouse button on objects when the cursor is a crosshair aligns them with the selected object (either vertically or horizontally). Pressing the left mouse button on the window background restores the original cursor.

attributes press right mouse button to display (and modify) the attributes of the object.

A few notes are in order. Not all objects can be resized. For example, the size of a radio button object is solely determined by the items that make up the radio button. All sizable objects have limits on how *small* they can be, and thus are not drawn smaller than their limits. The size of a scrollbar is constrained such that at any given time the length must be twice the width.

If the edges of two objects coincide, the edges do not appear on the VIB display, but they will be present when the application is prototyped or executed. However, objects should not overlap further than this.

VIB follows the standard Icon convention in which the upper-left corner of the window is at (0, 0), with the x coordinate increasing to the right and the y coordinate increasing in the downward direction.

The size of the generated interface window can be controlled by the *resize icon* that initially appears in lower-right corner of the canvas. This object can be dragged anywhere within the canvas via the left mouse button. Pressing the right mouse button over the object pops-up an attribute sheet describing its current dimensions. If the VIB window is resized such that it is smaller than the interface window, the interface window is automatically made smaller. If an object instance does not fall entirely within the bounds of the interface window, it will appear clipped in the generated program as well. That is, there is no automatic repositioning of objects to fit within the interface window.

Above the tool palette are three menus. The **File** menu provides the following functionality:

new creates a new VIB file.

open loads a saved VIB file.

save saves the VIB interface to the current file.

save as acts like *save* but prompts for a file name.

prototype writes a file with the VIB interface and a skeletal main program, then translates and executes it. Each callback generated by operating an object in the prototype window produces a line of output giving the vidget ID and the value. The prototype is exited by typing q in its window.

quit terminates the VIB session.

The **Edit** menu provides the following functionality:

copy makes a copy of the selected object.

delete deletes the selected object.

undelete restores the most recently deleted object.

align vert aligns objects with the x-coordinate of the selected object.

align horz aligns objects with the y-coordinate of the selected object.

The **Select** menu lists the ID fields of all the objects, allowing an object to be selected by name.

Keyboard shortcuts are indicated on the **File** and **Edit** menus for most items. Holding down the *meta* key and pressing the appropriate character is equivalent to choosing the menu item. (The *meta* key is a special shift key. It is sometimes labeled with a diamond or propeller or the word ALT.) The notation **delete @D** on the edit menu, for example, indicates that *meta-D* is the shortcut for delete.

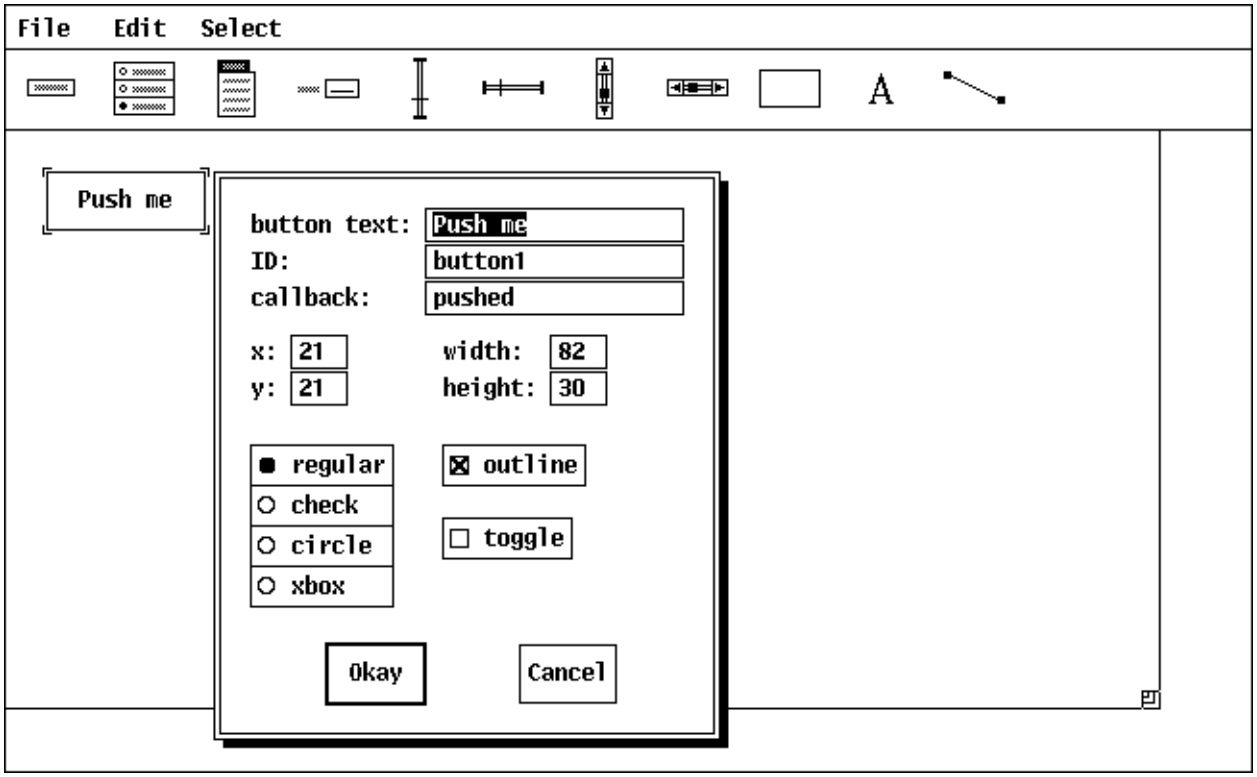
3. Attribute Sheets and Dialogues

VIB displays dialogue boxes to specify attributes, gather information and/or warn the user. The text fields of these pop-up windows can be edited; the tab key is used to move among them.

Attribute Sheets

As various objects are created, it usually is necessary to set attributes to customize the object to suit the needs of the application. The attribute sheet for an object is displayed by pressing the right mouse button upon the object. The editable features of an object are object-specific, but these usually include the x-y location of the object, the name of the object, a procedure to call when the object receives an event, and so forth. The Reference Section, appearing later in this document, describes the editable attributes of each object in particular.

Attribute sheets also contain *Okay* and *Cancel* buttons. Pressing either of these buttons makes the attribute sheet disappear: The Okay button applies changes to the object while the Cancel button does not. There is a thick line drawn around the Okay button to indicate that this is the default button of the attribute sheet; the Okay button can be selected by pressing the return key (as an alternative to using the mouse).



If the Okay button is selected VIB checks to ensure that entered values are valid. For example, if there is no value entered into the x-coordinate field, an error window appears describing this error. The error window disappears when either the *Dismiss* button is selected or the return key is typed. This in turn redisplays the attribute sheet until the error is corrected or the Cancel button is selected.

Dialogues

To open, save, or prototype a VIB interface, VIB displays a dialogue box requesting (or verifying) a file name for reading or writing, depending upon the nature of the request. If the *Okay* button is selected and the file name entered does not end in .icn, VIB appends the suffix to the value before attempting to access the file. If the file cannot be accessed (for example, the file does not exist for reading or the file cannot be written to), an error window appears describing the problem. The request can be canceled at any time by selecting the *Cancel* button.

4. The Application Interface

VIB supports the development of *event driven* applications where the central program loop is within the vidget library. When the application user manipulates an object on the screen, such as by clicking the mouse over a button, this event generates a *callback* to an Icon procedure associated with the object. The callback procedure for an object is provided separately by the programmer and its name is specified in the object's attribute sheet. The parameters depend on the type of object; details are contained in the reference guide that follows.

The interface specification edited by VIB is an Icon procedure ui(win,cbk) that passes a list of object specifications to the vidget library. The optional win parameter supplies an existing window on which the interface is to be created; if win is either null or a list of arguments for Window(), a new window is opened, and the new window is assigned to &window unless &window already has a window value. The optional cbk parameter provides a default callback procedure to be used for any object that does not provide one.

The ui procedure returns a table of vidgets. There is a vidget for each interface object, indexed in the table by the **id** name from the object's attribute sheet. There is also a root vidget that is the parent of all others. The user code that calls ui passes this root vidget to the procedure GetEvents() to enter the main event loop.

A typical event-driven application produced by VIB can be outlined as follows:

```
link vsetup

procedure main (args)
   vidgets := ui(args)
   process remaining arguments, open files, etc.
   sophisticated applications might add or modify vidgets here
   GetEvents(vidgets["root"], shortcuts)
end

procedure shortcuts(e)        (optional)
   process events not handled by vidgets
end

callback procedures
other procedures

VIB interface section — the procedure ui()
```

The skeletal procedure generated by VIB for new files follows a similar pattern. Applications that perform computations or update the display while waiting for events may require a different main event loop. See Reference 3.

5. Building Dialog Boxes

VIB can be used to construct dialog boxes for use in an application. VIB edits a single dialog box at a time. The code for each dialog box is kept in a separate source file and linked as part of the application. The application itself need not be constructed using VIB.

Dialog mode is configured using the attribute sheet of the canvas. This is called up by clicking the right-hand mouse button on the nested squares in the lower right corner. Checking the **dialog window** box enables dialog mode. The dialog box is named by entering a value in the **name** field; this value becomes the name of the generated dialog procedure.

A dialog box must contain at least one button that is not a toggle. Pressing such a button is the way the user dismisses a dialog box. A dialog box cannot contain a menu or region; VIB does not prohibit these objects, but they are ignored when the dialog box is actually created.

To use a dialog box, the application calls the procedure generated by VIB. The dialog box is then displayed, temporarily obscuring part of the application window. When the user presses a non-toggle button to dismiss the box, the dialog procedure returns the label of that button. Additionally, the global variable **dialog_value** is assigned a table containing the values of the objects in the dialog box. The table is indexed by object ID.

The signature of a dialog procedure is as follows:

```
procedure dialog_name(win, deftbl)
```

where win is the window in which the dialog is to appear and deftbl is an optional table of default values. If win is null, the subject window, &window, is used. Values in deftbl, which is indexed by object ID, provide initial defaults used when the dialog box is first displayed.

6. Reference Guide

Buttons

Buttons are control devices that are activated by pressing the mouse while over the button. They may appear in four different styles: regular, check box, circle, and xbox. An outline is optional. A button can be flagged as a toggle, which maintains a state of *on* or *off*. The attribute sheet of a button contains the following editable features:

text	is the label of the button. This may be an empty string.
id	is the name assigned to the object.
callback	specifies the procedure to call when the button receives an event. If no procedure is specified the event is essentially ignored.
x	specifies the x-coordinate of the upper-left corner of the button.
y	specifies the y-coordinate of the upper-left corner of the button.
width	specifies the width of the button.
height	specifies the height of the button.
style	specifies the look of the button. Four styles are supported: regular, check box, circle, and xbox.
outline	specifies whether an outline is to be drawn around the button.
toggle	specifies whether the button is a toggle button.

The callback procedure associated with a button is called when the mouse is pressed *and released* while over the button. If the mouse is released while off of the button, no event is sent to the application. The signature of button callbacks is as follows:

```
procedure button_cb(vidget, value)
```

where vidget is the actual button vidget created by VIB and value is the current value of the button. A regular button does not maintain a state and therefore its value is insignificant. However, if the button is a toggle, the button does maintain a state. A non-null value indicates that the button is set or on, while a null value indicates that the button is off.

Radio Buttons

Radio Buttons are collections of buttons in which exactly one button is set at any given time; an exception is the initial configuration in which no buttons are set. Selecting one button automatically unsets the previously highlighted button in the group. The attribute sheet of a radio button contains the following editable features:

id	is the name assigned to the object.
callback	specifies the procedure to call when the radio button receives an event. If no procedure is specified the event is essentially ignored.
x	specifies the x-coordinate of the upper-left corner of the radio button.
y	specifies the y-coordinate of the upper-left corner of the radio button.

The attribute sheet also contains *insert* and *delete* buttons. These are used to dynamically alter the number of entries in the radio button. Pressing on the insert button pops open a window querying the number of items to insert and where the insertion should take place. The default is to insert one item at the end of the list. Pressing on the delete button pops open a window querying the range of items to delete. The default is to delete the last item of the list.

The callback procedure associated with a radio button is called whenever one of its buttons is pressed (and the release takes place while over the button). If the mouse is released while off of the button, no event is sent to the application. The signature of radio button callbacks is as follows:

```
procedure radio_button_cb(vidget, value)
```

where vidget is the actual radio button vidget created by VIB and value is the current value of the radio button. The value of the radio button is the label of the currently highlighted button.

Menus

Menus are lists of buttons that appear temporarily on the screen and allow the user to select one item from a list; they can contain an arbitrary nesting of submenus. A menu appears when its *menu button* is pressed. A menu button is simply the visual representation of the menu that is visible when the menu is not active. Defining a menu therefore involves two parts: defining the text and position of the menu button, and defining the menu that is displayed as the result of pressing on the menu button. The attribute sheet of a menu provides the means to define the menu with submenus and choices. The editable features of a menu are as follows:

title	is the label that appears on the menu button.
id	is the name assigned to the object.
callback	specifies the procedure to call when a menu item is selected. If no procedure is specified the event is essentially ignored.
x	specifies the x-coordinate of the upper-left corner of the menu button.
y	specifies the y-coordinate of the upper-left corner of the menu button.

The attribute sheet also contains *insert* and *delete* buttons. These are used to dynamically alter the number of entries in the menu. Pressing on the insert button pops open a window querying the number of items to insert and where the insertion should take place. The default is to insert one item at the end of the list. Pressing on the delete button pops open a window querying the range of items to delete. The default is to delete the last item of the list.

When an item is added to a menu, a blank text field along with two buttons appear on the attribute sheet. The text field is used to define the label of the menu item, and the two buttons are used to define whether the menu item is a menu choice or a submenu. If the new item is to be a menu choice, all that needs to be done is to give the choice a label. If the new item is to be a submenu, mark the submenu box and click on the submenu button which will pop open a new window in which to define the submenu. VIB allows arbitrary nesting of submenus.

If a menu item is defined as a submenu, and then later the item is marked as a menu choice, this essentially deletes the predefined submenu (and all of its choices and submenus).

Once a menu has been defined, it can be viewed within VIB by pressing the middle mouse button on the menu button. This allows the appearance and behavior of the menu to be simulated without fully prototyping the interface. Pressing the right mouse button on the menu button again displays the attribute sheet of the menu object, providing a quick edit/simulation cycle.

The callback procedure associated with a menu is called when the menu has been displayed and the mouse is released while over one of its choices. The signature of menu callbacks is as follows:

```
procedure menu_cb(vidget, value)
```

where vidget is the actual menu vidget created by VIB and value is a list of labels defining the menu path of the selected choice. For example, if the menu has *open* and *close* as its choices and *open* is selected, ["open"] will be the value passed to the callback. If the menu has *font* as a submenu label and *Helvetica* as a choice within the submenu, then ["font", "Helvetica"] will be the value if *Helvetica* is selected. Thus choice names need not be unique across the entire menu, they can be distinguished by their path strings.

Text Input Fields

Text input fields are used to gather textual input from the user of the application. They consist of a label and a value; the value is editable and can accept a limited number of characters, as specified by the *max value length* attribute. The attribute sheet of a text input field contains the following editable features:

label is the label of the text input field.

value is the default value of the text input field.

id is the name assigned to the object.

callback specifies the procedure to call when the text input field receives an event (which happens when the return key is pressed and the text input field has the focus). If no procedure is specified the event is essentially ignored.

x specifies the x-coordinate of the upper-left corner of the text input field.

y specifies the y-coordinate of the upper-left corner of the text input field.

max value length specifies the maximum number of characters that the value can contain.

The callback procedure associated with a text input field is called whenever the return key is pressed. The signature of text input field callbacks is as follows:

```
procedure text_cb(vidget, value)
```

where vidget is the actual text vidget created by VIB and value is the current value of the text input field. The value of the object is the entered text string.

Sliders

Sliders are long rectangular buttons that display one or more scalar values as positions within a range. When the slidebar is clicked or dragged by the user, a scalar value is increased or decreased. Sliders can appear either vertically or horizontally. The attribute sheet of a slider contains the following editable features:

id is the name assigned to the object.

callback specifies the procedure to call when the slider receives an event. If no procedure is specified the event is essentially ignored.

filter filters out some of the events sent to the slider, if set. This corresponds to the non-continuous mode as described below.

left/top specifies the left value of the range for horizontal sliders or the top value of the range for vertical sliders. This can be a positive or negative value of type integer or real.

right/bottom specifies the right value of the range for horizontal sliders or the bottom value of the range for vertical sliders. This can be a positive or negative value of type integer or real.

x specifies the x-coordinate of the upper-left corner of the slider.

y specifies the y-coordinate of the upper-left corner of the slider.

length specifies the length of the slider.

width specifies the width of the slider.

The application can control the number of events passed to the slider. In *continuous* mode, the slider receives events as the slidebar is pressed, dragged, and released. In *non-continuous* mode, the slider receives a single event indicating the resulting value of the press-drag-release sequence. The slidebar can also be moved to a new location by clicking anywhere within the slider region. In both modes, this results in the generation of a single event. The signature of slider callbacks is as follows:

```
procedure slider_cb(vidget, value)
```

where vidget is the actual slider vidget created by VIB and value is the current numeric value of the slider.

Scrollbars

Scrollbars are sliders with a proportionally sized thumb capped on top and bottom by arrow buttons. The slidebar shows the current location of the window or associated device within some domain that is larger than available screen space and allows convenient random access; the arrow buttons allow more precise motion. Scrollbars can appear either vertically or horizontally. The attribute sheet of a scrollbar contains the following editable features:

id is the name assigned to the object.

callback specifies the procedure to call when the scrollbar receives an event. If no procedure is specified the event is essentially ignored.

filter filters out some of the events sent to the scrollbar, if set. This corresponds to the non-continuous mode as described below.

left/top specifies the left value of the range for horizontal scrollbars or the top value of the range for vertical scrollbars. This can be a positive or negative value of type integer or real.

right/bottom specifies the right value of the range for horizontal scrollbars or the bottom value of the range for vertical scrollbars. This can be a positive or negative value of type integer or real.

x specifies the x-coordinate of the upper-left corner of the scrollbar.

y specifies the y-coordinate of the upper-left corner of the scrollbar.

length specifies the length of the scrollbar.

width specifies the width of the scrollbar.

The application can control the number of events passed to the scrollbar. In *continuous* mode, the scrollbar receives events as the slidebar is pressed, dragged, and released. In *non-continuous* mode, the scrollbar receives a single event indicating the resulting value of the press-drag-release sequence. The slidebar can also be moved to a new location by clicking anywhere within the scrollbar region. In both modes this results in the generation of a single event. Additionally, the scrollbar is called when the increment/decrement buttons are pressed. The signature of scrollbar callbacks is as follows:

```
procedure scrollbar_cb(vidget, value)
```

where vidget is the actual scrollbar vidget created by VIB and value is the current numeric value of the scrollbar.

Regions

Regions are rectangular areas. The application can safely draw in these areas and receive uninterpreted events. (While there is nothing to prevent an application from drawing anywhere within the window, it is not recommended: The application might mar the display by drawing over vidgets.)

The attribute sheet of a region contains the following editable features:

id is the name assigned to the object.

callback specifies the procedure to call when the region receives an event. If no procedure is specified the event is essentially ignored.

x specifies the x-coordinate of the upper-left corner of the region.

y specifies the y-coordinate of the upper-left corner of the region.

width specifies the width of the region.

height specifies the height of the region.

line width specifies the thickness of the region outline. If zero, the region is not outlined.

Regions differ from other user interface objects in that the events sent to the region's callback procedure are uninterpreted. This provides the application with flexibility. The signature of the callback is as follows:

```
procedure region_cb(vidget, e, x, y)
```

where vidget is the vidget that VIB created, e is the Icon event code, and x and y are the mouse coordinates at the time of the event.

Labels

Labels consist of read-only text. They do not receive events and therefore do not have callbacks associated with them. The attribute sheet of a label contains the following editable features:

text	is the text to display on the screen.
id	is the name assigned to the object.
x	specifies the x-coordinate of the upper-left corner of the label.
y	specifies the y-coordinate of the upper-left corner of the label.

Lines

Lines are provided to decorate the interface. They can be arbitrarily thick and may appear solid or dashed. Lines do not receive events and therefore do not make use of callback procedures. The attribute sheet of a line, which appears by clicking the right mouse button upon the line object, contains the following editable features:

id	is the name assigned to the object.
line width	specifies the width of the line.
x1	specifies the x-coordinate of endpoint one.
y1	specifies the y-coordinate of endpoint one.
x2	specifies the x-coordinate of endpoint two.
y2	specifies the y-coordinate of endpoint two.
style	specifies whether the line is solid or dashed.

7. Converting XIB Applications

VIB's original ancestor was the X-Icon Interface Builder, XIB. It functioned similarly but created files in a different format. The Icon library program uix translates the specifications contained in an XIB application, prototype, or specification file into VIB form. uix takes a single command argument naming its input file; if no argument is supplied, it reads standard input. The generated VIB application is written to standard output.

The output from uix is a functional prototype application that contains all the objects (buttons, sliders, etc.) from the input file but none of the user Icon code. This must be added manually. Note that the newly generated ui procedure returns a *table* of vidgets rather than just the root vidget returned in XIB applications.

8. Acknowledgements

As XIB and then VIB have evolved, they have benefited greatly from the comments of Ralph Griswold, Clint Jeffery, Jon Lipp, Ken Walker, and Yarko Tymciurak.

This work was supported in part by the National Science Foundation under Grant CCR-8901573.

References

1. R. E. Griswold and M. T. Griswold, *The Icon Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, NJ, second edition, 1990.
2. C. L. Jeffery, G. M. Townsend and R. E. Griswold, *Graphics Facilities for the Icon Programming Language; Version 9.0*, The Univ. of Arizona Icon Project Document IPD255, 1994.
3. J. Lipp, *Window Interface Tools for Version 9 of Icon*, The Univ. of Arizona Icon Project Document IPD259, 1994.
4. M. Cameron, *XIB: X-Icon Interface Builder*, The Univ. of Arizona Tech. Rep. 92-34, 1992.