Window Interface Tools for Version 9.0 of Icon

Jon Lipp

IPD259

Abstract

This document describes a user interface toolkit available to Icon programmers using the graphics facilities in Version 9.0 of Icon. The toolkit extends the basic window model by providing for layout of and event routing to virtual input/output devices attached to windows.

June 23, 1994

Department of Computer Science
The University of Arizona
Tucson, AZ 85721

Introduction

[Jeff94] describes the graphics facilities of Version 9.0 of Icon provided in the form of built-in functions and extensions to Icon's file data type. These facilities are easy to use for simple windowing applications. However, modern graphical user interface technology requires a more sophisticated approach. In particular, the complexity of modern interfaces motivates more robust models of windows as I/O devices as well as a more robust collection of implementation techniques.

This document describes a library of user input facilities built on top of Icon. The library consists of a collection of canonical user interface devices available to Icon programmers.

Vidgets

A *vidget* is defined as a "virtual input/output device gadget", and is used as the basic descriptive label for the tools described herein. At the very basic level, a vidget is a rectangular region of a window defined by an (x, y) coordinate pair for the upper left corner, and a width and height for the dimensions. Most vidgets handle events passed to them and are responsible for their own visual representation. Many vidgets accept their own events to perform various actions.

At the Icon level, the data structure adopted for maintaining the internals of a vidget is a record. The fields of these records is described below under **The fields of a vidget record.**

The root frame and composition of vidgets

There are basically two classes of vidgets, simple vidgets themselves, and vidget *frames*. A frame acts as an organizational entity that contains a collection of vidgets within its rectangular region, and dispatches events that fall within its domain to the appropriate vidget. Most varieties of frames are hidden from the user within wrapper procedures provided by the library. A vidget is usually told to draw itself by its parent frame.

Since vidgets must then be placed within a vidget frame, at the top level of this hierarchy is the *root frame*. This is a special frame that always encompasses the entire window area, and is able to handle events such as a window resize.

User Interface Primitives

A number of garden-variety user interface devices have been written and are provided by the library as finished products. This section describes some of these devices. Most devices except the very simplest buttons and toggles are examples of vidget *composition*; that is, they are constructed by laying out a collection of simpler vidgets within an enclosing *frame*.

Buttons are the simplest form of input devices, a portion of the screen used not to display information but to implement user-driven control flow. They vary in size, shape, appearance, and operation. Buttons are activated (pressed) by locating the mouse pointer in the portion of the screen in which the button appears, and clicking the mouse.

Toggles are buttons that retain their state between mouse events; instead of each mouse click activating the operation associated with the toggle, mouse clicks turn the toggle *on* and *off*.

Radio buttons are collections of toggles composed in such a way that at any time no more than one button is selected. Toggles in radio buttons are related by an exclusive-or constraint. When a toggle is selected, any other toggle in the radio button is deselected.

Menus are collections of buttons that appear temporarily on the screen and allow the user to select one or more items from a list.

Menu bars are collections of buttons that typically appear across the top or bottom of a region of a window. Menu bars usually are present on-screen most of the time, as visual reminders of the various menus available. Clicking on a button in a menu bar pulls down an associated menu.

Sliders are combination input/output devices. They are long rectangular buttons that graphically display one or more scalar values as positions in some range. When the slidebar is clicked or dragged by the user, a scalar value is increased or decreased. The scalar value is then used by the program.

Scrollbars are examples of vidget *composition*: sliders with proportionally sized thumbs are capped on either end by arrow buttons. The slidebar shows the current location of the window or associated device within some display that is larger than available screen space and allows convenient random access; the arrow buttons allow more precise motion (such as moving up or down a single line). The size of the thumb is determined by the ratio of the window size to the total size of the region being scrolled through. As this area may be variable in size (as in text editors), the thumb then changes size in accordance with the varying ratio.

Dialog boxes are further examples of device composition. A dialog box is used to obtain several related input values from the user. Each of the input values may be specified either by keyboard input, by a slidebar, or by some other device appropriate to the value. These techniques can be used in combination. For example, the approximate value can be given by a mouse click, and the value can then be edited by keystrokes. Buttons are used to specify when the dialog box can be closed.

Couplers are objects that describe relationships between devices, application data sets, or both. Couplers do not have an on-screen presence in and of themselves, but rather serve as "glue" that binds screen objects to underlying data. Their primary purpose is to send the flow of control to any callback procedure associated with a particular vidget.

How Vidgets Communicate

Different applications and different programmers have different ideas about the level at which communication with the interface should take place. The levels supported in this library are event translation, procedure callbacks, and couplers.

Event translation — VEvent(frame, e) is used to process individual events through a vidget frame. This method of communication ususally is not useful for an individual vidget, as most accept events using Event() in their internal event loops. Used on a procedure, this function attempts a lookup based on (x, y), and returns the results of the event loop of the vidget found. If no vidget accepts the event, VEvent() fails.

The result of a vidget's event loop is specific to the vidget. A vidget fails if passed an irrelevant event, as for example, if a character from the keyboard is passed to a button vidget. This is sufficient for simple applications that wish to utilize standard flow of control.

Procedure callbacks — the vidget can be created with a callback procedure. The callback procedure is passed two parameters: the Icon record that defines the vidget and a value corresponding to some internal state or value associated with the vidget. For example, Vradio_buttons passes to its procedure callback the vidget record itself and the string label of the button selected. For a Vslider the callback is passed the slider record and the scalar value of the slider position. A standard callback procedure heading looks like

```
procedure callback(vidget, value)
```

Coupler Variables — A coupler is an object that holds one (or more) values and a list of clients. A vidget utilizes a coupler by setting the coupler to a value, which in turn notifies any vidgets, procedures, or other couplers on the coupler's client list. If the client is another vidget, then the couplerset() procedure associated with that client is called, and the parameters are the client, the caller (the vidget who set the coupler) and the value of the coupler. If the client is a procedure, then the parameters are as described above; the vidget record sent is the caller specified in the VAddClient() call used to add the procedure to the client list of the coupler, and the value is simply the value of the coupler. If the client is another coupler, then that coupler is set using the same parameters passed to the original coupler.

When a vidget is created, it constructs its own internal coupler that resides in its callback field. Any clients passed to the vidget through the callback field upon creation of the vidget are automatically added as clients to the vidget's internal coupler. This allows the implicit creation of links between vidgets without manually creating couplers and using VAddClient(). See the examples at the end of this document of how to use couplers and the callback field. There are different couplers, depending on the way the value is manipulated: Vcoupler, Vrange_coupler, and Vbool_coupler.

Thus, when a vidget is created, the callback field may contain either a single callback client, which may be a coupler, vidget, or procedure, or several clients in the form of an Icon list to be added to the vidget's internal coupler. The creation procedure for the vidget detects the list and read the clients from it.

The fields of a vidget record

Since a callback procedure receives as its first parameter the record associated with a vidget, it is necessary to know the field names of the vidget to reference in order to access certain information. All vidgets (with the exception of coupler variables and Vline) have the following field names:

win	the window binding
ax, ay	the absolute coordinates in relation to the binding in field win
aw, ah	the width and height in pixels
callback	an internal coupler
id	a user-assigned id of any Icon data type
uid	a unique integer assigned internally upon creation

A Vline has the following field names, corresponding to the absolute values of its endpoints: ax1, ay1, ax2, and ay2.

All couplers share the fields:

value	the value of the coupler
id	an indentifier, set internally to the constant V_COUPLER
uid	a unique integer assigned internally upon creation

Using the Library

In order to access the library, it is necessary to link certain files. All programs using the library must link the file vidgets. To use the vidgets listed, the appropriate file must be linked:

link file	vidgets
vbuttons	Vbutton, Vtoggle, Vcheckbox, Vline, Vmessage
vdialog	Vdialog
vgrid	Vgrid
vmenu	Vsub_menu, Vmenu_bar, Vpull_down_pick_menu
vradio	Vhoriz_radio_buttons, Vvert_radio_buttons
vscroll	Vhoriz_scrollbar, Vvert_scrollbar
vslider	Vhoriz_slider, Vvert_slider
vidgets	Vpane, Vframe, Vroot_frame

The file vidgets also contains links to the coupler vidgets, GetEvents(), and various other internal procedures. For example, to use Vbutton, Vvert_radio_buttons, and Vvert_slider in a program, use the following link declaration:

```
link vidgets, vbuttons, vslider, vradio
```

Almost all functions and objects in the library are identified by a leading capital V. Creating a vidget is as simple as invoking the creation procedure, which entails passing several parameters. For example, to create a two-dimensional button with label Push Me, located within the root frame root at coordinates (10, 50), on window win, which calls the procedure CallBack(), with id 1, with default width and height, and with no outline:

```
Vbutton(root, 10, 50, win, "Push Me", CallBack, 1, V_2D_NO)
```

The library allows the capability to implicitly or explicitly insert the vidget into the root frame, and the above code is equivalent to

```
button := Vbutton(win, "Push Me", CallBack, 1, V_2D_NO)
VInsert(root, button, 10, 50)
```

This latter option may be useful if a vidget is created but is not immediately inserted into the root frame at a specific location. An example is a dialog box, which may be positioned at a context-specific location.

In addition to creating vidgets, several actions may be performed on an individual vidget. These are VDraw(), VErase(), VOutline(), VResize(), VRemove(), VInsert(), and VEvent(). In practice, vidgets are created and then the root frame manages all these tasks. Descriptions of the functions are in the **User's Reference.**

Event Handling

An event handler is a continuous loop that reads events from the window and dispatches them in various ways. The library provides an event handler called GetEvents() described in the **User's Reference.** A simple template for an event handler is:

```
repeat {
    e := Event(win)
    if e === "q" then stop()
    return_value := VEvent(root, e, &x, &y)
    write(return_value)
    }
```

This loop sends events to the root frame, lets it route the events onto any vidgets, which then execute their event loops and return a value. The VEvent call to root fails if (&x, &y) is not accepted by any vidget in root.

Coordinate Systems

The origin of any vidget's coordinate system is based upon the window binding with which it was created . The virtual origin of a window binding can be changed, via the dx, dy attributes for windows. Therefore, any output via a vidget's window binding specified in its w field is relative to this origin. For most vidgets these dx, dy attributes are (0, 0), but for some vidgets, it may be useful to change them to the coordinates given when inserted into the frame. This is the case for a Vpane, if the user wants to create a "virtual sub-window" on the screen at a certain region. For example, to create a region at coordinates (50, 50) with width and height of 200:

```
subwin := XBind(win)
WAttrib(subwin, "dx=50", "dy=50")
Clip(subwin, 0, 0, 200, 200)
region := Vpane(root, 50, 50, subwin, callback, id, 200, 200)
```

Any drawing operations to this vidget then have an origin of (0, 0) defined at (50, 50) on subwin and output is clipped by a 200x200 region. For example,

```
DrawRectangle(region.win, 10, 10, 300, 300)
```

draws a portion of a rectangle with the upper left corner at (60, 60) in the main window, but at translated coordinates (10, 10) in the subwindow region. The output is clipped within the region, so only the top and left sides of the rectangle are drawn.

The absolute (x, y) coordinates of a vidget in relation to the coordinate system of root are stored in the fields ax and ay of the vidget.

A vidget is inserted into a frame by VInsert() using an (x, y) coordinate pair. If the coordinate is a positive integer, or *absolute* coordinate, the vidget is positioned relative to the upper-left corner of the parent frame. If the coordinate is a negative integer, or an *offset* coordinate, then the vidget's lower-right corner is placed relative to the lower-right corner of the frame. For example,

```
VButton(root, -20, -10, win, "On")
```

inserts a button whose right side is 20 pixels from the right side of the frame, and whose bottom edge is 10 pixels from the bottom of the frame. Offset and absolute coordinates can be mixed between the x and y coordinates. Similarly,

```
VButton(root, -20, 10, win, "On")
```

places a button in the upper-right corner of the frame. Since the library requires a negative integer to detect offset positioning, it is not possible to position a vidget at the extreme lower-right corner of a frame. It can be positioned at 1 pixel off, however. For example,

```
VButton(root, -1, -1, win, "On").
```

If a vidget is inserted with a real-valued coordinate, or *normalized* coordinate, this must be a number between 0.0 and 1.0, and specifies a position in the frame as a percentage of the dimension of the frame. For example,

```
VButton(root, 0.25, 0.50, win, "On")
```

places a button at 25% from the left side, and 50% down from the top.

Button Styles

Buttons and toggles can be created with one of three predefined styles, specified in the style field of the vidget: 2-d, check box, and circle box. These are denoted by the symbolic constants V_2D, V_CHECK, and V_CIRCLE. These indicate the button has an outline. To specify a style without an outline, use the constants V_2D_NO, V_CHECK_NO, and V_CIRCLE_NO.

Dialog Boxes

A dialog box is a temporary vidget frame that is opened on the window via the procedure VOpenDialog(). A dialog contains a collection of vidgets that are either *inserted* or *registered* into the dialog using VInsert() or VRegister() respectively. Registered vidgets contain a value that can be set and changed by the user. The value is displayed when the dialog box is opened, and the updated value is returned when the dialog box is closed. Inserted vidgets do not have a value associated with them; they merely serve as control vidgets. Values are passed to and from registered vidgets via a list ordered by their id fields. The structure of a dialog box is subject to several restrictions:

- To pass data values, a vidget must be *registered* using VRegister() instead of VInsert().
- The vidgets that can be registered with a dialog box are Vtoggle, Vvert_slider, Vhoriz_slider, Vvert_radio_buttons, Vhoriz_radio_buttons, Vtext, Vvert_scrollbar, and Vhoriz_scrollbar.
- A Vbutton with id of V_OK *must* be inserted into the dialog box using VInsert(). This enables the dialog box to close itself and pass back the data values.
- Vidgets must be registered or inserted relative to the frame with absolute coordinates only.

Control buttons (such as "Ok" or "Cancel") are inserted (not registered) because they do not pass values back through the return list. These buttons are assigned special constants for their id fields of V_OK and V_CANCEL. Values of registered vidgets are set by passing a list of values into the call to VOpenDialog(). The items in the list are placed in the registered vidgets in the sorted order of the vidget's id fields. If the button with id field V_OK is pressed, the dialog box is closed, and a list of altered values is returned. Otherwise, if the button with id field V_CANCEL is pressed, the dialog closes, and the original list is returned. A string label can be passed to VOpenDialog() through the default_string parameter. This sets a control button to be the default button activated upon pressing the return key while the dialog box is open. The tab key is used to move through any Vtext vidgets in the dialog box, and the order is determined by the id fields of the vidgets. A sample dialog box is made as follows:

```
db:= Vdialog(win, 30, 30)

VRegister(db, Vtext(win, "Name : ", ,1), 0, 0)
VRegister(db, Vtext(win, "Address : ", ,2), 0, 50)
VRegister(db, Vtext(win, "Phone : ", ,3, 3, &digits++'()-' ), 0, 100)
```

```
VRegister(db, Vvert_radio_buttons(win, ["student", "faculty"] , ,4), 0, 150)
VInsert(db, Vbutton(win, " Ok ", ,V_OK), 50, 300)
VInsert(db, Vbutton(win, "Cancel", ,V_CANCEL), 150, 300)

VFormat(db)
```

This builds a dialog box db with three Vtext input fields, one set of Vvert_radio_buttons, and two control buttons, Ok and Cancel. To open this dialog,

```
data_list := ["Joe Isuzu", "Gould-Simpson Bldg", , "student"]

VOpenDialog(db, &x, &y, data_list, " Ok ")
```

initializes the Name field with "Joe Isuzu", the Address field with "Gould-Simpson Bldg", and the Vradio_buttons initially set to "student". The default button to be pressed upon hitting the return key is the " Ok " button.

Note that using a dialog box in this fashion does not require making a Vroot_frame or doing a VResize().

Scrollbars

As described above, scrollbars are used to represent the position of a window region within some larger space, and the thumb size is determined by the ratio of window size to the total region size being scrolled through. To specify this relation to the scrollbar vidget, The window_size field is given as a scalar value lying within the range of [bottom_bound, top_bound]. The value of window_size must be given in the same units as the bounding variables. For example, to create a scrollbar that represents scrolling through a 20-line-document using a 5 line "view" window:

```
Vvert_scrollbar(win, callback, id, length, width, 20, 0, 1, 5)
```

This makes a scrollbar with the value at the top of the scrollbar as 1, at the bottom, 20, with an increment of 1, and a window size of 5.

A Simple Demonstration

This section contains an example of the basic steps required to utilize the vidget library.

```
link vidgets, vbuttons

procedure main()
local win, root, toggle
   win := open("demo", "x")
```

The first vidget created in any user interface must be the root frame.

```
root := Vroot_frame(win)
```

Upon creating the Vroot_frame, other vidgets can be inserted into it using an explicit call to VInsert(), or implicitly upon creation of the vidget (as explained above).

```
Vbutton(root, 10, 30, win, "Button One", cb, 1)
toggle := Vtoggle(win, "Toggle Two", cb, 2)
VInsert(root, toggle, -10, 30)
```

After the vidgets have been inserted into the frame,

```
VResize(root)
```

is called. This procedure sets the absolute sizes of all the vidgets, clears the window area, and signals all vidgets to draw themselves. Alternatively, VResize() can be called immediately after creation of the root frame, but any subsequent vidgets must be told explicitly to draw themselves. Either method requires the root frame to be resized first before any vidgets contained within can be drawn, as the vidgets' internal absolute coordinate systems are not set until their root's absolute coordinate are first set by VResize(). An event loop can now be started.

```
    GetEvents(root, quit)
end

procedure quit(e)
    if e === "q" then stop()
end

procedure cb(vid, val)
    write(vid.id, " ", val)
end
```

Since the user-assigned identifier is stored internally in the vidget's record field id, it is necessary for the callback procedure to reference the field to obtain the value.

This demonstration puts two buttons on the window, then lets the user press each one, which in turn calls a callback procedure cb(), until a q is pressed outside either vidget.

(Note: In parameter declarations, [frame, x, y,] *indicates that these parameters are optional in the function call. If included, the vidget is inserted into* frame *at* x, y.)

GetEvents(frame, MissedEvents, AllEvents, ResizeEvents)

GetEvents(...) handles events from the window associated with frame using Event(). The functionality of the procedure is thus: If a lookup on &x, &y in frame fails, then the event is passed to an optional procedure passed through the parameter MissedEvents. If the lookup succeeds, the event is sent to the vidget and processed accordingly. In either case, the event is passed to the procedure indicated by AllEvents. If a resize event occurs, ResizeEvents will be called before any other procedure.

Parameters:

frame	a vidget of type Vframe or Vroot_frame
MissedEvents	an optional procedure to handle events from Event() in the event the lookup fails; passed the parameters e, &x, &y, where e is the value returned by Event()
AllEvents	an optional procedure to handle all events; params are the same as for MissedEvents
ResizeEvents	an optional procedure to handle resize events; the parameter passed is the root frame passed to GetEvents()

VAddClient(coupler, client, caller)

VAddClient(...) adds client to the callback list of coupler. If client is a procedure, then the procedure is passed the record associated with caller and the value of the coupler.

Parameters:

coupler	a coupler vidget
client	a vidget or a procedure
caller	see above

VDraw(vidget)

VDraw(...) instructs vidget to draw itself.

VErase(vidget)

VErase(...) instructs vidget to erase itself (the rectangular region on its window binding defined by its width and height).

VEvent(vidget, e, x, y)

VEvent(...) executes the event loop of vidget.

Parameters:

vidget	the vidget
e	the event code
x, y	absolute coordinates of the event relative to the window

VFormat(frame)

VFormat(...) computes the minimum width and height for frame that would encompass all vidgets inserted into it. This routine requires that all vidgets have been inserted into frame using absolute coordinates. Upon computing the bounds, VFormat() assigns these values to the aw and ah fields of frame, thus bypassing a need for the user to compute the bounds manually. This routine is useful when frame is of type Vdialog. This procedure must be called *before* frame is inserted into its parent frame, otherwise the bounds are not recorded by the parent frame, and further resizings of the window do not maintain the settings by VFormat().

VInsert(frame, vidget, x, y)

VInsert(...) inserts vidget into frame.

Parameters:

frame	a frame to insert into
vidget	the vidget to insert
x, y	the coordinates in frame to insert the vidget

VOpenDialog(dialog, x, y, data, default_string)

VOpenDialog(...) opens the dialog box dialog at x, y. The dialog box is "smart" in that it does not open up with any part of it off the edge of the window. Thus, no coordinate checking is required beforehand. However, if the window is too small for the dialog box, then it gets pushed off the top.

Parameters:

dialog	a vidget of type Vdialog
x, y	the position of the upper left corner of the dialog box relative to the window
data	a list of data values corresponding to the vidgets registered with dialog
default_string	a string label of the control button to press upon hitting return

Defaults:

x, y	calculated to center the dialog box in the window
data	[] (an empty list); all data fields reset to default
default_string	no default button is specified

VReformat(vidget, length, width)

VReformat(...) changes the size of a scrollbar vidget. This is used on resize events. (See the demo xscroll.icn at the end of this document for an example of how VReformat() is used.)

Parameters:

vidget	a vidget of type Vvert_scrollbar or Vhoriz_scrollbar
length	the new length of the scrollbar
width	the new width of the scrollbar

VRegister(dialog, vidget, x, y)

VRegister(...) registers vidget with the dialog box dialog as an editable vidget at position (x, y). This means that this vidget holds a value editable by the user, as opposed to control buttons like OK or CANCEL, which have no value associated with them. A requirement by dialog boxes is that all insertions must be made with absolute coordinates.

Parameters:

dialog	a vidget of type Vdialog()
vidget	a vidget of a type listed above in section **Dialog Boxes** (page 7)
x, y	the absolute coordinates in the dialog to insert the vidget

VRemove(frame, vidget, no_erase)

VRemove(...) removes vidget from frame. vidget is automatically erased. If this is not desired, set the no_erase field to a non-null value.

Parameters:

frame	a frame to remove from
vidget	a vidget to remove
no_erase	non-null to not erase the vidget after removal

VResize(vidget, x, y, w, h)

VResize(...) sets the absolute coordinates of vidget. If vidget is a frame, this sets the absolute coordinates of all vidgets inserted in the frame based on the coordinates provided. If vidget is the root frame, this draws all vidgets. For all other vidgets, this procedure is called automatically by the frame it is contained in to set the absolute position and size based on the virtual size passed to the vidget in creation. This procedure *must* be called on the root frame prior to invoking any event handler or performing a VDraw().

Parameters:

vidget	the vidget to resize
x, y	the absolute coordinates in frame to insert the vidget
w, h	the absolute pixel width and height of the vidget

VSet(vidget, val)

VSet(...) sets the state of vidget to val. For example, Vradio_buttons has a string label value associated with it, and Vtoggle has a null or non-null value. If vidget is a coupler, then the value associated with the coupler is set accordingly. If the coupler is a Vbool_coupler, then val is ignored, and the boolean coupler value is set to non-null.

Parameters:

vidget	a vidget or coupler
val	a value to set the coupler to

VToggle(coupler)

VToggle(...) toggles the value associated with the boolean coupler coupler. This function only works on a Vbool_coupler.

Parameters:

coupler	a coupler variable of type Vbool_coupler

VUnregister(dialog, vidget)

VUnregister(...) takes a vidget off the registered list of editable vidgets for dialog. This does not erase the vidget, as it can only be called after a dialog has been closed. Thus, the vidget being removed is not visible.

Parameters:

dialog	a vidget of type Vdialog()
vidget	a vidget of a type listed above in section **Dialog Boxes** (page 7)

VUnSet(coupler)

VUnSet(...) sets the value associated with the boolean coupler coupler to &null. This function only works on a Vbool_coupler.

Parameters:

coupler	a coupler variable of type Vbool_coupler

The following entries are vidget creation procedures.

Vbutton([frame, x, y,] w, s, callback, id, style, aw, ah)

Vbutton(...) creates a button vidget. VEvent() returns the id field of the button on a successful press.

Parameters:

w	a window file
s	a string label for the button
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
style	a symbolic constant indicating the style of the button
aw	the width in pixels of the button
ah	the height in pixels of the button

Defaults:

style	V_2D
aw	8 + the pixel width of s
ah	8 + the font height of w

Vcheckbox([frame, x, y,] w, callback, id, size)

Vcheckbox(...) creates a check-box vidget. The functionality of this vidget is identical to a Vtoggle().

Parameters:

w	a window file
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
size	the width and height in pixels of the vidget

Defaults:

size	15

Vdialog(w, padx, pady)

Vdialog(...) creates a frame for a popup dialog box. Vidgets such as Vtext_in and Vradio_buttons are inserted using VRegister(). An example of creating a dialog box is shown in the program listings at the end of this document. When vidgets are inserted into the dialog, the point (0, 0) is considered to be the upper-left corner of the vidget plus any padding in the x or y dimension. For example,

```
dialog := Vdialog(win, 30, 30)
VRegister(dialog, Vtext(win), 0, 0)
```

places a Vtext vidget at (30, 30) in dialog.

Parameters:

w	a window file
padx	number of pixels for vertical border padding between vidgets and dialog outline
pady	number of pixels for horizontal border padding between vidgets and dialog outline

Defaults:

padx	20
pady	20

Vframe([frame, x, y,] w, aw, ah)

Vframe(...) creates a vidget frame into which other vidget objects may be "inserted" using VInsert(). aw and ah are commonly set after vidgets have been placed within, and the bounds can be figured.

Vgrid([frame, x, y,] win, callback, id, aw, ah, rows, cols)

Vgrid(...) creates a grid vidget with pixel width of aw and pixel height of ah. The number of grid divisions is determined by rows and cols. Events are passed to the callbacks associated with the Vgrid in the manner described above, with the exception that the value passed to the callback is a three element list consisting of [row, col, e], where row and col are the grid element the event occurred in, (the upper left corner grid element numbered as (0, 0)), and e is the event that occurred.

Parameters:

w	a window file
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
aw	the width in pixels of the grid
ah	the height in pixels of the grid
rows	the number of vertical rows in the grid
cols	the number of horizontal columns in the grid

Defaults:

aw, ah	100, 100
rows, cols	10, 10

Vhoriz_radio_buttons([frame, x, y,] w, data, callback, id, style)

Vhoriz_radio_buttons(...) creates a frame containing radio buttons positioned horizontally with labels corresponding to each entry in the list passed as s. VEvent() returns the string label of the button selected. If a callback is specified, the value passed is the string label of the radio button pressed.

Parameters:

w	a window file
data	a list of string labels
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
style	a button style

Defaults:

callback	&null (no clients called)
id	&null (no identifier is sent to the procedure)
style	V_2D

Vhoriz_scrollbar([frame, x, y,] w, callback, id, length, width, left_bound, right_bound, increment, window_size, discontinuous)

Vhoriz_scrollbar(...) creates a horizontal scrollbar. VEvent() returns the value of the internal coupler variable associated with the scrollbar upon release of the mouse.

Parameters:

w	a window file
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
length	the length (width) of the scrollbar
width	the pixel width (height) of the scrollbar
left_bound	the value limit at the left of the scrollbar
right_bound	the value limit at the right of the scrollbar
increment	the increment value used by the arrow buttons
window_size	the size of the view window, in the same units as the range
discontinuous	if non-null, callbacks are invoked only upon *release* of the thumb

Defaults:

width	15 pixels
left_bound	0.0
right_bound	1.0
increment	10% of the range

Vhoriz_slider([frame, x, y,] w, callback, id, length, width, left_bound, right_bound, init, discontinuous)

Vhoriz_slider(...) creates a horizontal slider. VEvent() returns the value of the internal coupler variable associated with the slider upon release of the mouse.

Parameters:

w	a window file
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
length	the length (width) of the slider
width	the pixel width (height) of the slider
left_bound	the value limit at the left of the slider
right_bound	the value limit at the right of the slider
init	initial value of the slider
discontinuous	if non-null, callbacks are invoked only upon *release* of the slider

Defaults:

width	15 pixels
left_bound	0.0
right_bound	1.0
init	bottom_bound

Vline(w, x1, y1, x2, y2)

Vline(...) creates a line. This line vidget can be inserted into the root frame, and is automatically drawn along with other vidgets upon invocation of VResize(root). This vidget does not accept events. The two coordinate pairs can be specified with absolute, normalized, and/or offset positions as described above. The (x, y) coordinate pair is not specified in the VInsert(). For example, to insert a line into the root frame from (100,100) to (200, 200):

```
VInsert( root, Vline(win, 100, 100, 200, 200))
```

To insert a line from the middle of the screen to the lower right corner (to support resize events):

```
VInsert( root, Vline(win, 0.50, 0.50))
```

Parameters:

w	a window file
x1, y1	coordinates of first point
x2, y2	coordinates of second point

Defaults:

x1, y1	0
x2	window width
y2	window height

Vmenu_bar([frame, x, y,] w, s, menu, s, menu, ...)

Vmenu_bar(...) creates a menu bar consisting of the strings passed as the 2nd, 4th, etc. arguments, which call the menus defined beforehand by Vsub_menu() in the following argument. An example of building a hierarchical menu system is contained in the program listings at the end of this document. VEvent() returns the result of the callback associated with the menu choice made; if there is no callback specified, the list of string labels of the path to the menu choice is returned (See Vsub_menu below). If the mouse is not released on a menu selection, VEvent() fails.

Parameters:

w	a window file
s	a string label
menu	a submenu defined by Vsub_menu()

Defaults:

menu	&null (no menu is called)

Vmessage([frame, x, y,] w, s)

Vmessage(...) creates a simple text message vidget. This does not handle events, and is only implemented to provide convenience in placing text on the screen.

Parameters:

w	a window file
s	a string label for the message

Vpane([frame, x, y,] w, callback, id, linewidth, aw, ah)

Vpane(...) creates a vidget that consists of a region on the screen. It acceps events and processes them normally via its callback, but it has no visual representation aside of an outline, specified by the value of linewidth. VEvent() returns the id field of the vidget.

Parameters:

w	a window file
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
linewidth	the linewidth of the outline; if null, no outline
aw	width in pixels of the pane
ah	height in pixels of the pane

Defaults:

linewidth	null (no outline)

Vpull_down_pick_menu([frame, x, y,] w, s, callback, id, size, centered)

Vpull_down_pick_menu(...) creates a vidget that displays the results of a pick made from a pull down list of string entries from the list s. VEvent() returns the string label of the choice selected.

Parameters:

w	a window file
s	a list of string labels for the choices
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
size	number of characters in the display field
centered	non-null if the items in the pull down are to be centered

Defaults:

size	24

Vroot_frame(w)

Vroot_frame(w) creates a frame that encompasses the whole window. It is necessary for the operation of the vidget library.

Parameters:

w	a window file

Vsub_menu(w, s, callback, ...)

Vsub_menu(...) creates a menu that lies underneath the top level of menus created using Vmenu_bar() above. When an entry in a Vsub_menu is selected, the callback is passed two parameters: the record data structure of the menu bar entry the sub menu lies under, and a one-item list containing the string label of the sub menu entry. For multiple level sub menus, the list passed as the value to the callback is a "path" of string labels from the top submenu to the menu choice. *Note: the callback field may not contain another vidget or a coupler variable; only a callback procedure.*

Parameters:

w	a window file
s	a string label corresponding to the first entry in the menu
callback	a procedure to be called by the menu item associated with s

(The last two parameters are repeated for each menu entry in the sub_menu.)

Defaults:

callback	&null (no procedure is called)

Vtext([frame, x, y,] w, s, callback, id, sz, mask)

Vtext(...) creates a textual input line. VEvent() returns &null if the operation on the vidget succeeds, the symbolic constant V_NEXT if tab, down arrow, or return key is pressed, or the constant V_PREVIOUS if the up arrow key is pressed. In addition, when the return key is pressed, the callback associated with the vidget is called. If the linefeed key is pressed, this calls the callback of the Vtext and returns the result. This is used in dialog boxes, when a Vtext vidget needs to notify its callback but does not want to indicate that the return key was pressed. (See vidgets.icn at the end of this document for an example of how the callback is utilized with a Vtext vidget.)
The cursor may be positioned by clicking the pointer within the data field. Text can be blocked out by dragging the mouse over the text field. The blocked-out text is then considered to be under the cursor, and is subject to be edited as one character. For example, blocking out all of the text and hitting backspace deletes all characters. If a is pressed, the letter a replaces all the blocked text. To block out all of the data, position the cursor at the end of the data and press the mouse button. To initialize the Vtext with data upon creation, use the code "\\=" within the prompt field s to indicate the rest of the string is data. For example, if s = "pages=\\= 10", the resulting prompt is "pages=" and the initial data is "10". To set the data after the Vtext vidget has been inserted into a frame, use VSet(vidget, string).
Full line editing capabilities are via the following commands:

left arrow	move left
right arrow	move right
backspace	delete to the left of the cursor

For several Vtext vidgets linked together via a dialog box:

up arrow	move to previous line
tab, return, or down arrow	move to next line

Parameters:

w	a window file
s	a string prompt
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
sz	maximum number of characters in text field
mask	a cset of allowable characters to be inputted

Defaults:

s	""
sz	18
mask	&cset (all characters are allowed)

Vtoggle([frame, x, y,] w, s, callback, id, style, aw, ah)

Vtoggle(...) creates a toggle vidget. VEvent() returns the id field of the toggle button on a successful press.

Parameters:

w	a window file
s	a string label for the toggle
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
style	a symbolic constant indicating the style of the toggle
aw	the width in pixels of the button
ah	the height in pixels of the button

Defaults:

style	V_2D
aw	8 + the pixel width of s
ah	8 + the font height of w

Vvert_radio_buttons([frame, x, y,] w, data, callback, id, style)

Vvert_radio_buttons(...) creates a frame containing radio buttons positioned vertically with labels corresponding to each entry in the list passed as L. VEvent() returns the string label of the button selected. If a callback is specified, the value passed is the string label of the radio button pressed. This vidget may also be created using Vradio_buttons(...).

Parameters:

w	a window file
data	a list of string labels
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
style	a button style

Defaults:

callback	&null (no clients called)
id	&null (no identifier is sent to the procedure)
style	V_2D

Vvert_scrollbar([frame, x, y,] w, callback, id, length, width, bottom_bound, top_bound, increment, window_size, discontinuous)

Vvert_scrollbar(...) creates a vertical scrollbar. VEvent() returns the value of the internal coupler variable associated with the scrollbar upon release of the mouse.

Parameters:

w	a window file
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
length	the length (height) of the scrollbar
width	the pixel width of the scrollbar
bottom_bound	the value limit at the bottom of the scrollbar
top_bound	the value limit at the top of the scrollbar
increment	the increment value used by the arrow buttons
window_size	the size of the view window, in the same units as the range
discontinuous	if non-null, callbacks are invoked only upon *release* of the thumb

Defaults:

width	15 pixels
bottom_bound	0.0
top_bound	1.0
increment	10% of the range

Vvert_slider([frame, x, y,] w, callback, id, length, width, bottom_bound, top_bound, init, discontinuous)

Vvert_slider(...) creates a vertical slider. VEvent() returns the value of the internal coupler variable associated with the slider upon release of the mouse.

Parameters:

w	a window file
callback	a single callback or list of callbacks as described above
id	a user assigned id of any Icon data type
length	the length (height) of the slider
width	the pixel width of the slider
bottom_bound	the value limit at the bottom of the slider
top_bound	the value limit at the top of the slider
init	initial value of the slider
discontinuous	if non-null, callbacks are invoked only upon *release* of the slider

Defaults:

width	15 pixels
bottom_bound	0.0
top_bound	1.0
init	bottom_bound

The following entries are coupler vidgets.

Vbool_coupler (value)

Vbool_coupler(...) creates a boolean coupler variable with an initial value of value. The value can be null to indicate the coupler is unset, or non-null to indicate it is set. Client vidgets or callback procedures are registered with the coupler via VAddClient(), and the value of the coupler is altered via VToggle(), VSet(), VUnSet(), or by the vidgets themselves.

Parameters:

value	a value to initialize the coupler to

Default:

value	&null

Vcoupler (value)

Vcoupler(...) creates a coupler variable with an initial value of value. Client vidgets or callback procedures are registered with the coupler via VAddClient(), and the value of the coupler is altered via VSet() or by the vidgets themselves.

Parameters:

value	a value to initialize the coupler to

Default:

value	&null

Vrange_coupler(min, max, value, inc)

Vrange_coupler(...) creates a coupler that stores numeric values within a range specified by min and max. The field inc is used by tools that increment the value by a certain value, for example, the arrows in a scrollbar composition. If both min and max are specified as integer values, then the value of the coupler is truncated to an integer. Otherwise the value is real.

Parameters:

min	minimum bound of the coupler
max	maximum bound of the coupler
value	the initial value
inc	an increment

Defaults:

min	0.0
max	1.0
value	min
inc	10% of the range

Acknowledgements

The design and implementation of the library is the result of many helpful comments, recommendations, and discussions by members of the Icon Project, including Mary Cameron, Ralph Griswold, Clint Jeffery, Gregg Townsend, and Ken Walker. Gregg Townsend has made a number of improvements to the initial library.

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

Sample Programs

Scroll

```
# This program provides a window within which to scroll an image
#

link options
link vidgets, vscroll
link wopen
link xcompat

global win, im_win, view_width, view_height
global scv, sch

procedure main(args)
    local opts, file, scrollbar_width, picw, pich, root

    opts := options(args, "f:w+h+")
    file := \opts["f"] |
        stop("Usage: scroll -f file [-w window size/width] [-h window height]")
    view_width := \opts["w"] | 300
    view_height := \opts["h"] | view_width
    scrollbar_width := 15
#
# Load in the bitmap; get the dimensions.
#
    im_win := WOpen("canvas=hidden", "image=" || file) |
        stop("Couldn't make temporary bitmap.")
    picw := WAttrib(im_win, "width")
    pich := WAttrib(im_win, "height")

    win := WOpen("label=" || file, "size=" ||
        (view_width + scrollbar_width + 1) || "," ||
        (view_height + scrollbar_width + 1) ) |
            stop("*** cannot open file")

    root := Vroot_frame(win)
#
# Create two scrollbars.
#
    scv := Vvert_scrollbar(root, -1, 0, win, sl_cb, 1,
```

```
        view_height,scrollbar_width, pich, 0, , view_height)
    sch := Vhoriz_scrollbar(root, 0, -1, win, sl_cb, 2, view_width,
     scrollbar_width, 0, picw, , view_width)

    VResize(root)
#
# Draw the initial view of the pixmap, based on the scrollbar's values.
#
    sl_cb(scv, scv.callback.value)
    sl_cb(sch, sch.callback.value)
#
# Now get events, pass control to the procedure quit() if an event is not
# captured by a vidget.
#
    GetEvents(root, quit, , resize)
end

#
# Terminate the program on a keypress of "q".
#
procedure quit(e)

    if e === "q" then stop("End scroll.")
end

procedure resize(root)

    VReformat(scv, WAttrib(scv.win, "height") - 15)
    VReformat(sch, WAttrib(sch.win, "width") - 15)
end

#
# Copy a portion of the bitmap to the main
# window based on the values of the scrollbars.
#
procedure sl_cb(caller, val)
    static vpos, hpos
    initial vpos := hpos := 0

    (caller.id = 1, vpos := val) | hpos := val
    CopyArea(im_win, win, hpos, vpos, view_width, view_height, 0, 0)
```

end

Splat

```
# This program draws randomly sized circles in random colors at random places
#

link color, vidgets, vradio, wopen

global COLORS, state, region, Go, Stop, Quit, Redraw

procedure main (arg)
   local win, root, rb

   Go := "GO"; Stop := "STOP"; Quit := "QUIT"; Redraw := "CLEAR"
   COLORS := ["plum", "slate blue", "olive drab", "coral", "wheat"]

   win := WOpen("label=splat", "size=500,500") |
      stop("*** cannot open window")
   root := Vroot_frame(win)
   rb := Vradio_buttons(root, 10, 10,
      win, ["GO", "STOP", "CLEAR", "QUIT"], state_change)
   region := Vpane(win, , , 2)
   VInsert(root, region, 80, 10, 400, 400)

   VSet(rb, "STOP")
   VResize(root)

   state := Stop
   repeat {
      while (*Pending(win) > 0) | (state == Stop) | (state == Redraw) do
         VEvent(root, Event(root.win), &x, &y)
      Splat(region)
      }
end

procedure state_change(vid, val)
   state := val
   if state == Quit then exit()
   if state == Redraw then
      EraseArea(region.win, region.ax + 1, region.ay + 1, region.aw - 2,
         region.ah - 2)
end
```

```
procedure Splat(box)
    local x, y, w, h, c, y3

    h := w := ?100
    x := ?(box.aw – w)
    y := ?(box.ah – h)
    y3 := box.ay + ?(box.ah)
    c := ?5
    Shade(box.win, COLORS[c])
    FillArc(box.win, box.ax + x, box.ay + y, w, h)
end
```

Vdemo

```
# This program demonstrates a variety of vidgets
#

link dialog, options, vidgets, vdialog, vmenu, vradio, vbuttons, vtext
link vscroll

global dialog

procedure main(args)
    local opts, font, win, ht, title, wid, row, pad, root, cv, i, ti, scb
    local tm, s, max, rb1, rb2, rb3
    local FontTypeSubMenu, FontSubMenu, CompMemSubMenu, ExecMemSubMenu
    local FileMenu, OptionsMenu

    opts := options(args, "f:wh")
    font := \opts["f"]
    wid := \opts["w"]
    ht := \opts["h"]
    title := "All the vidgets."



    win:= WOpen("label=" || title,
        "size=" || (\wid | 550) || "," || (\ht | 550)) |
            stop("*** can't open window")

    pad := WAttrib(win, "fheight")+10
    row := []
    every i := 0 to 9 do put(row, i*pad)

    root := Vroot_frame(win)

## Vdialog
    dialog:= Vdialog(win)

    VRegister(dialog, Vtext(win, "Button Text: ", ,1), 0, row[1])
    VRegister(dialog, Vtext(win, "Button Id : ", ,2), 0, row[2])
    VRegister(dialog, Vtext(win, "Callback : ", ,3), 0, row[3])
    VRegister(dialog, Vtext(win, "X: ", ,4,3, &digits), 10, row[5])
```

```
    VRegister(dialog, Vtext(win, "Y: ", ,5,3, &digits), 10, row[6])
    VRegister(dialog, Vtext(win, "W: ", ,6,3, &digits), 100, row[5])
    VRegister(dialog, Vtext(win, "H: ", ,7,3, &digits), 100, row[6])
    VRegister(dialog, Vradio_buttons(win, ["solid", "onoff"] , ,8), 200, row[5])
#
# Attach a slider to a textual input device.
#
    cv := Vcoupler()
    VAddClient(cv, ti := Vtext(win, , cv, 9, 3, &digits))
    VAddClient(cv, scb := Vvert_scrollbar(win, cv, 10, 75, , 0, 100, 1))
    VInsert(dialog, scb, 275, row[5])
    VRegister(dialog, ti, 300, row[5])
#
# Control buttons.
#
    Vbutton(dialog, 100, row[8], win, " Ok ", ,V_OK)
    Vbutton(dialog, 200, row[8], win, "Cancel", ,V_CANCEL)

    VFormat(dialog)

    Vmessage(root, 10, 0.5, win, "Press mouse button to open a dialog.")

## Vsub_menu, Vmenu_bar
#
# Have to create the menu system bottom-up, so... start at the deepest leaves.
#
# Use Vsub_menu to build sub-menus.
#
# Once the sub-menus have been built, use Vmain_menu to make the menu bar.
#
    FileMenu := Vsub_menu(win,
        "New", m_cb,
        "Open", m_cb,
        "Close", m_cb,
        "Save", m_cb,
        "Save As", m_cb,
        "Print", m_cb,
        "--------", ,
        "Quit", exit
        )
```

```
FontTypeSubMenu := Vsub_menu(win,
      "Normal", m_cb,
      "Bold", m_cb,
      "Italic", m_cb,
      "Underline", m_cb,
      )

FontSubMenu := Vsub_menu(win,
      "Times", FontTypeSubMenu,
      "Courier", FontTypeSubMenu,
      "Palatino", FontTypeSubMenu,
      "Schoolbook", FontTypeSubMenu,
      "Helvetica", FontTypeSubMenu,
      "Symbol", m_cb,
      "Arial", FontTypeSubMenu,
      "Sans Serif", FontTypeSubMenu,
      )

CompMemSubMenu := Vsub_menu(win,
      "Constant Table Size", m_cb,
      "Field Table Size", m_cb,
      "Global Symbol Table Size", m_cb,
      "Identifier Table Size", m_cb,
      "Local Symbol Table Size", m_cb,
      "Line Number Space", m_cb,
      "String Space", m_cb,
      "File Name Table Size", m_cb,
      )

ExecMemSubMenu := Vsub_menu(win,
      "Block Region", m_cb,
      "String Region", m_cb,
      "Evaluation Stack", m_cb,
      "Co-expression Blocks", m_cb,
      "Qualifier Pointer Region", m_cb,
      )

OptionsMenu := Vsub_menu(win,
      "Font", FontSubMenu,
      "Font Size", m_cb,
      "--------", ,
```

```
        "Parameter String", m_cb,
        "Library Folders", m_cb,
        "Compiler Memory", CompMemSubMenu,
        "Execution Memory", ExecMemSubMenu
        )

    tm := Vmenu_bar(root, 0, 0, win,
        "File", FileMenu,
        "Options", OptionsMenu
        )

## Vline
    VInsert(root, Vline(win, , tm.ah, , tm.ah))

    Vbutton(root, 10, 40, win, "Push Me", popup, "This is a notice button.")

## Vpull_down_pick_menu
    s := ["Times", "Helvetica", "NewCentury", "Symbol", "Palatino",
          "Zapf Chancery"]
    max := 0
    every i := !s do max <:= *\i
    Vpull_down_pick_menu(root, 200, 40, win, s, pd_cb, "pull-down", max+1)

## Vradio_buttons
    rb1:= Vhoriz_radio_buttons(root, 10, 0.70, win,
       ["Here", "is", "a", "list", "of", "radio", "buttons"],
       rb_cb, 1)
    rb3:= Vradio_buttons(root, -10, -10, win,
       ["Here", "is", "a", "list", "of", "radio", "buttons"],
       rb_cb, 2, V_CIRCLE)

    VSet(rb1, "list")

    VResize(root)
    GetEvents(root, PopUpDialog)
end

procedure PopUpDialog(e)
    local i, nl
    static data
    initial data := ["one", "", "nothing", 23, 67, 12, 23, "solid", 17]
```

```
    if e === "q" then stop()
    if e === (&lpress |& mpress | &rpress) then {
        nl := VOpenDialog(dialog, &x, &y, data, " Ok ")
        every i := !nl do write(i)
        data := nl
        }
end

procedure pd_cb(vid, val)
    Notice(vid.win, "Chose: " || val)
end

procedure rb_cb(vid, val)
    write(vid.id, ": ", val)
end

procedure m_cb(vid, val)
    write("\nmenu bar: ", vid.s)
    writes("choice : ")
    every writes(!val, " ")
end

procedure popup(vid)
    Notice(vid.win, vid.id)
end
```

References

[Jeff94] Jeffery, C. L., Gregg M. Townsend, and Ralph E. Griswold. Graphic Facilities for the Icon Programming Language; Version 9.0. IPD255, Department of Computer Science, University of Arizona, 1994.

Vidget Library Index

procedure	page
GetEvents()	10
VAddClient()	10
VDraw()	11
VErase()	11
VEvent()	11
VFormat()	11
VInsert()	12
VOpenDialog()	12
VReformat()	12
VRegister()	13
VRemove()	13
VResize()	13
VSet()	14
VToggle()	14
VUnregister()	14
VUnSet()	14

vidget	page
Vbutton()	15
Vcheckbox()	15
Vdialog()	16
Vframe()	16
Vgrid()	17
Vhoriz_radio_buttons()	18
Vhoriz_scrollbar()	19
Vhoriz_slider()	20
Vline()	21
Vmenu_bar()	22
Vmessage()	22
Vpane()	24
Vpull_down_pick_menu()	23
Vroot_frame()	24
Vsub_menu()	24

vidget	page
Vtext()	25
Vtoggle()	26
Vvert_radio_buttons()	26
Vvert_scrollbar()	27
Vvert_slider()	28

coupler	page
Vbool_coupler()	29
Vcoupler()	29
Vrange_coupler()	30