XIB: X-Icon Interface Builder

Mary Cameron

TR 92-34

Abstract

This document describes XIB, a user interface builder used to specify and generate graphical user interfaces for X-Icon. XIB allows quick prototyping of user interface designs: Designs can easily be created, tested, and modified in a “what you see is what you get” environment. To provide maximum flexibility with respect to tool and interface creation, interfaces can be generated with or without an underlying tool. XIB is essentially a front-end for the Icon WIT library.

December 9, 1992

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

Introduction

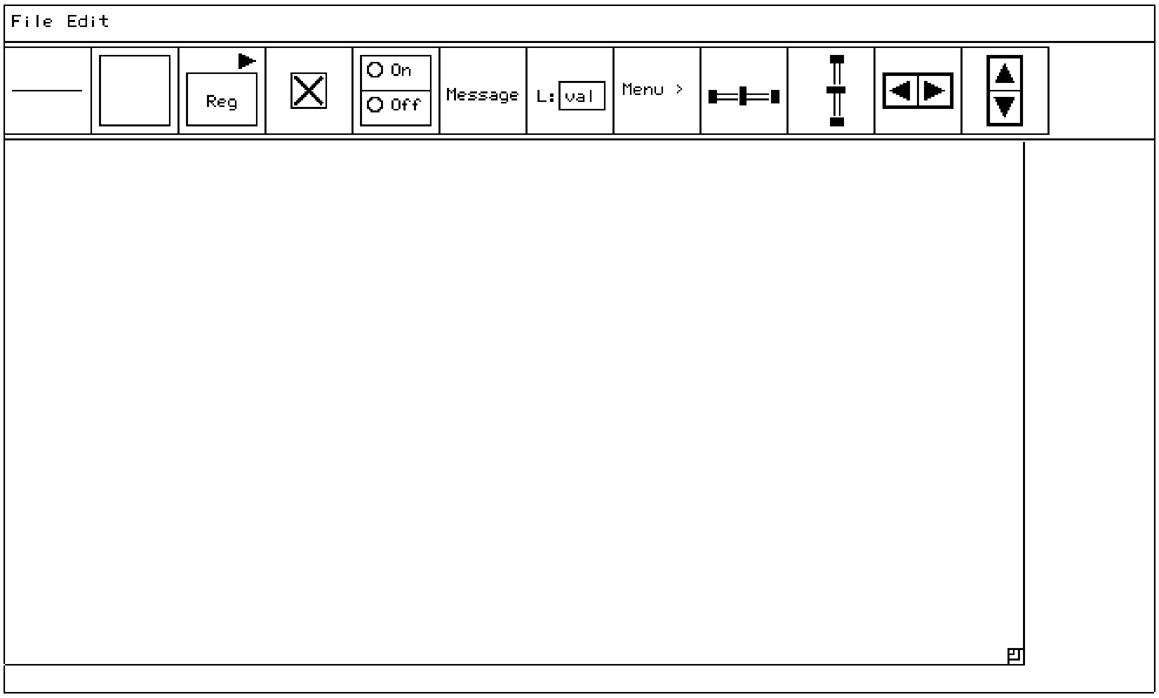
XIB is a user interface builder that enables rapid prototyping of tool interfaces for X-Icon [Jef92]. It provides a work area and a palette of user interface objects, instances of which can be created and customized within the work area. Interfaces can be prototyped and/or saved, generating code that is either stand-alone or to be included with the underlying tool. Building interfaces then becomes a question of taste and style as opposed to patience.

When an interface is prototyped or saved, XIB generates an Icon source program containing calls into the Icon WIT library [Lip92]. If the interface is to be prototyped, then a complete program is generated such that it can be executed. If the interface is only to be saved, then a procedure is generated which can then be included with the underlying tool. In this way, interfaces can be created stand-alone and then later incorporated into the tool; the transition from prototype-mode to tool-mode is relatively painless.

The functionality provided by the Icon WIT library is more extensive than what is presented by XIB. Thus advanced or complex interfaces may need to use the library directly, using XIB only as a stepping stone. It is therefore advantageous to be familiar with the WIT library as well as X-Icon.

XIB Overview

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



The palette contains the following objects:

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

Regions are rectangular areas turned over to the application for both input and output purposes. 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 distort the display by drawing over WIT objects, and the application will not be informed of end user activities.

Buttons are control devices that are activated by pressing the mouse while over the button. Buttons may appear in many different styles: regular, check box, check box without an outline, circle, and circle without an outline. The default style is regular; this can be changed by selecting a different style from the pop-up menu that appears when the right mouse button is pressed on the button palette image. Selecting a new default updates the palette image to the selected button style and creates button instances thereafter in this new style.

Check Boxes are switch buttons that are marked when *on* and blank when *off*. They differ from check-style buttons in that the size of the check box can be arbitrarily large, and there is no accompanying text.

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

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

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

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

Sliders 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. Sliders can be either vertical or horizontal.

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 virtual area 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.

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.
resize	drag left mouse button upon a corner of the object. The opposite corner is anchored.
copy	press *c* within the canvas or select *copy* from the **edit menu**.

delete press *d* within the canvas or select *delete* from the **edit menu**.

front press *f* within the canvas or select *front* from the **edit menu**.

back press *b* within the canvas or select *back* from the **edit menu**.

align press *v* or *h* within the canvas or 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.

XIB follows the standard X 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. *Front* and *back* functions are provided to control z-axis ordering of objects. Although it is possible to specify overlapping objects within XIB, it is not recommended.

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 XIB 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.

In addition to the tool palette, there are two menus provided that are displayed in response to mouse activities. The first is the **file menu** which provides the following functionality:

open a saved XIB file.

save XIB interface to a file, which generates a file containing a procedure named `ui` that creates WIT objects according to their XIB specifications. Appended to the end of the file is *XIB code* which allows the file to be re-edited in XIB via *open*. **It is important not to delete or alter this code as it may inhibit the re-editing of an XIB interface!**

save as similar to the *save* option except that a new file name is requested.

prototype similar to the *save* option except that callback templates and a main procedure are generated such that the program is complete; the complete program is then compiled and executed.

quit terminate the XIB session.

Thus after an interface editing session takes place, the interface can be saved (via the *save* option) and later re-edited (via the *open* option). An interface can be loaded from the command line upon XIB start-up. The syntax is

```
xib <file name>
```

where file name is a file previously generated by XIB. If the given file name does not end in .icn, XIB appends the suffix before attempting to access the file. If the file does not exist or cannot be read, an error window appears describing the problem.

The other menu is the **edit menu**, which provides the following functionality:

copy	makes a copy of the selected object.
delete	deletes the selected object.
front	brings the selected object to the front of the canvas.
back	pushes the selected object to the back of the canvas.
redraw	redraws the screen.
clear	deletes all objects.
align vert	aligns objects with the x-coordinate of the selected object.
align horz	aligns objects with the y-coordinate of the selected object.

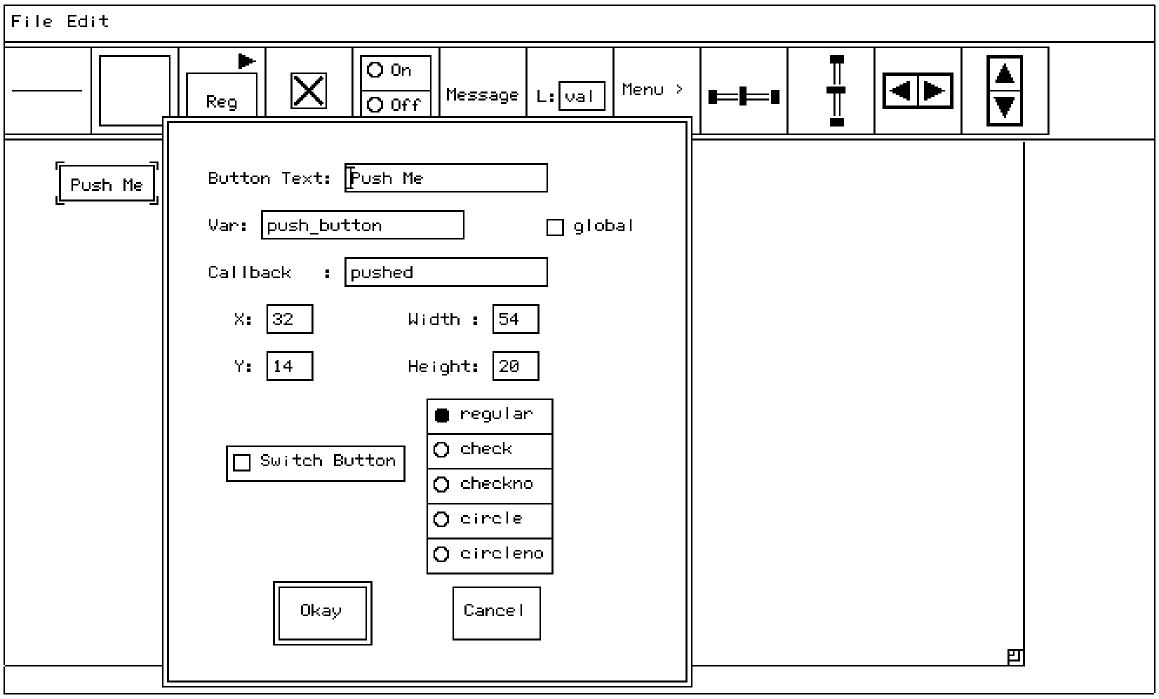
XIB Attribute Sheets and Dialogues

XIB 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 variable 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 double 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 XIB does error checking 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 if 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 an XIB interface, XIB 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, then XIB 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.

Event Handling

XIB supports the development of *event driven applications*. Event driven applications typically provide a graphical interface to the end user and then *respond* to user interactions. In XIB this amounts to creating user interface objects and associating callback procedures with them. For example, if an application creates a button, then whenever that button is pressed the button's callback is notified, allowing the application to respond in the appropriate manner. Callback procedures are, in fact, the sole means of connecting a tool with its interface.

Some user interface objects provided by XIB do not receive events (and therefore are not really user interface objects in the traditional sense). These include lines and messages, which are provided for display purposes only. Thus lines and messages do not have callbacks associated with them.

The majority of XIB's supported objects do receive events; these events trigger callback procedures which are defined as follows:

```
procedure callback(wit, value)
```

where wit is the WIT object that XIB created and value is the current value of the object. For example, if a text field is created and the end user types in a value followed by return, the callback procedure associated with the text field is called. The wit is then the WIT text field object and the value is the text entered into the field.

There is one exception to the above protocol, and that involves regions. Regions are areas in which the application has complete control over what is displayed as well as the semantics of end user activities. The application might need to know when and where mouse clicks occur, whether or not keys are pressed, and so forth. Regions effectively allow the application to work at the X-Icon level. The signature of region callbacks is as follows:

```
procedure callback(wit, e, x, y)
```

where wit is the WIT object that XIB created, e is the X-Icon event code, and x and y are the mouse coordinates (relative to the upper-left corner of the region) at the time of the event.

Interface Generation

Once an interface has been created by XIB, it can be either saved or prototyped. Both generate a file containing the procedure ui followed by *XIB code*. The procedure takes an X-Icon window as a parameter (supplying one if necessary), creates a WIT root frame with WIT objects inserted into it, and returns the root frame. The ui procedure effectively creates the visual image as specified within XIB. The XIB code allows the editing session to be reconstructed. It consists of Icon comments that must remain undisturbed.

Prototyping an interface, in addition to generating the ui procedure and XIB code, generates a main procedure and callback templates such that the file is a complete Icon program. Selecting the prototype option thus generates, compiles, and executes the interface from within XIB. To quit the prototype, simply type q on the prototype window (the mouse must not be positioned over a WIT object). The advantage of prototyping is obvious: it allows one to see the results quickly and with minimal effort. However, it is important to note that prototyping an interface completely rewrites the file; thus if semantics are added to callback procedures and the interface is re-prototyped to the same file, those enhancements will be overwritten.

Saving an interface simply generates the ui procedure and the XIB code. The idea is that the main procedure and callback procedures are supplied by the application in separate files. Thus an interface can be edited and re-saved such that callback procedures are not overwritten. Saving an interface does require that the complete program be compiled, linked, and executed outside of XIB.

The recommended approach to interface development is to first prototype the interface. This provides a quick feel for the appearance of WIT objects as well as their behavior. The generated callback templates print out event information each time the routine is called, so event triggering can be monitored. It also might be worthwhile to view the generated program.

Once the prototype is satisfactory, the main procedure and callback procedures should be moved into a separate file, and the interface should be *saved* from here on out, affecting only the ui procedure and XIB code.

Reference Guide

Lines

Lines are provided to decorate the interface. They consist of two coordinates and can be drawn arbitrarily thick. Additionally, they 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:

var	is the variable name of the WIT object that XIB generates.
global	defines the var to be global, if marked.
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.

Regions

Regions are rectangular areas. The application can safely draw in these areas and receive uninterpreted events. Using regions does require a knowledge of X-Icon, as X-Icon provides the necessary drawing primitives as well as event code semantics. The attribute sheet of a region contains the following editable features:

var	is the variable name of the WIT object that XIB generates.
global	defines the var to be global, if marked.
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 null, 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(wit, e, x, y)

where wit is the WIT object that XIB created, e is the X-Icon event code, and x and y are the mouse coordinates (relative to the upper-left corner of the region) at the time of the event. The coordinate (0, 0), then, refers to the upper-left corner of the region.

Regions also differ from other objects in that they receive an initial callback after all objects have been created but before any user-generated events occur. This allows the application to, for example, display an image within the region that the end user can then manipulate. The event code passed to the callback is the &resize event as defined within X-Icon.

Buttons

Buttons are control devices that are activated by pressing the mouse while over the button. They may appear in many different styles: regular, check box, check box without an outline, circle, and circle without an outline. They can also be created as a switch or toggle in which the button maintains a state of either *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.

var is the variable name of the WIT object that XIB generates.

global defines the var to be global, if marked.

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.

switch specifies whether the button is a switch button. Switch (or toggle) buttons maintain a state of *on* or *off*, as opposed to regular buttons.

style specifies the look of the button. Currently, five varying styles are supported: regular, check box, check box-no outline, circle, and circle-no outline.

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(wit, value)

where wit is the actual WIT button object created by XIB 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 switch, 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.

XIB creates WIT button objects such that the id field of the WIT object is the label of the button. Thus the same callback procedure can be used for various buttons: it can determine which button was pressed by accessing the wit.id value.

Check Boxes

Check boxes are switch buttons that are marked when *on* and blank when *off*. They differ from check-style buttons in that the size of the check box can be arbitrarily large, and there is no accompanying text. The attribute sheet of a check box contains the following editable features:

var is the variable name of the WIT object that XIB generates.

global defines the var to be global, if marked.

callback specifies the procedure to call when the check box 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 check box.

y specifies the y-coordinate of the upper-left corner of the check box.

size specifies the length and width of the check box.

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

```
procedure check_box_cb(wit, value)
```

where `wit` is the actual WIT check box object created by XIB and `value` is the current value of the check box. A non-null value indicates that the check box is set or on, while a null value indicates that it 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:

var is the variable name of the WIT object that XIB generates.

global defines the var to be global, if marked.

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(wit, value)

where wit is the actual WIT radio button object created by XIB and value is the current value of the radio button. The value of the radio button is the label of the currently highlighted button.

Messages

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

text	is the text to display on the screen.
var	is the variable name of the WIT object that XIB generates.
global	defines the var to be global, if marked.
x	specifies the x-coordinate of the upper-left corner of the message.
y	specifies the y-coordinate of the upper-left corner of the message.

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.
var	is the variable name of the WIT object that XIB generates.
global	defines the var to be global, if marked.
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(wit, value)

where wit is the actual WIT text object created by XIB and value is the current value of the text input field. The value of the object is the entered text string.

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.

var is the variable name of the WIT object that XIB generates.

global defines the var to be global, if marked.

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. XIB 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 XIB 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(wit, value)
```

where `wit` is the actual WIT menu object created by XIB 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.

Sliders

Sliders are long rectangular buttons that graphically 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:

var is the variable name of the WIT object that XIB generates.

global defines the var to be global, if marked.

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(wit, value)
```

where `wit` is the actual WIT slider object created by XIB 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:

var is the variable name of the WIT object that XIB generates.

global defines the var to be global, if marked.

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(wit, value)
```

where wit is the actual WIT scrollbar object created by XIB and value is the current numeric value of the scrollbar.

Acknowledgements

The design and functionality of XIB evolved from numerous group discussions with Icon Project members, including Ralph Griswold, Clinton Jeffery, Jon Lipp, Gregg Townsend, and Ken Walker. XIB has tremendously benefited from their knowledge and insights on user interface designs and aesthetics, postulations of the "It would be nice if ..." variety, and their black and white approach to *features* versus *bugs*.

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

References

[Jef92] Clinton L. Jeffery. X-Icon: An Icon Windows Interface. Technical Report 91-1d, Department of Computer Science, University of Arizona, July 1992.

[Lip92] Jon Lipp. Window Interface Tools for X-Icon. Technical Report 92-32, Department of Computer Science, University of Arizona, December 1992.