User's Guide

CMU/SEI-91-UG-4
September 1991

Serpent: Dialogue Editor User's Guide

User Interface Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

The Software Engineering Institute is not responsible for any errors contained in these files or in their printed versions, nor for any problems incurred by subsequent versions of this documentation.

The Software Engineering Institute is not responsible for any errors contained in these files or in their printed versions, nor for any problems incurred by subsequent versions of this documentation.

Table of Contents

List of Figures

1 Introduction

This guide introduces the Serpent dialogue editor, which is used to communicate the dialogue between the end user and the functional, or application, portion of a system. This dialogue specifies both the presentation of application information and end-user interactions.

Put differently, the dialogue editor provides an environment in which to develop and maintain Slang dialogue specifications. Most of what is built in the editor translates directly into Slang, the Serpent dialogue specification language. That is, the Slang code generated from the dialogue editor can be compiled and run to produce the user interface for an application.

Although the dialogue editor hides much Slang syntax, users of the editor should understand the conceptual model of Slang, as well as the syntax of “code snippets” and expressions that are used to define much of the interactive behavior. See *Serpent: Slang Reference Manual* for information about Slang.

1.1 This Manual

This manual is a guide for using the dialogue editor. Readers are assumed to have read and understood the concepts described in *Serpent Overview* and *Serpent: Slang Reference Manual.*

This manual also assumes that readers are running the Serpent system with the Motif window manager (MWM) and the instructions for moving and resizing windows are based on the capabilities of MWM. If another window manager is used, these instructions may not pertain; the method of moving and resizing windows will be specific to that other window manager.

1.1.1 Organization

This guide includes the following sections:

- **Introduction.** Chapter 1 contains an introduction and the conventions for this manual.
- **Getting Started.** Chapter 2 contains the instructions for building a user interface that allows a user to increment a counter.
- **Menubar Example.** Chapter 3 contains the instructions for building a pull-down menu.

- **Command Descriptions.** Chapter 4 describes individual dialogue editor commands.

1.1.2 Typographical Conventions

The following conventions are observed in this manual:

Buttons	**Bold**
Code examples	`Courier typeface`
Command widgets	**Bold**
Menu names	**Bold**
Menu items	`Courier typeface`
Syntax	`Courier typeface`
Toolbox icons	**Bold**
Variables, attributes, etc.	`Courier typeface`
Warnings and cautions	***Bold, Italics***

The dialogue editor is designed to be used with a three-button mouse. These buttons are referred to as “left,” “right,” and “middle.”

1.2 Other Serpent Documents

The following documents provide information about the Serpent system:

Serpent Overview
Introduces the Serpent system.

Serpent: System Guide
Describes installation procedures, specific input/output file descriptions for intermediate sites and other information necessary to set up a Serpent application.

Serpent: Saddle User’s Guide
Describes the language that is used to specify interfaces between an application and Serpent.

Serpent: Slang Reference Manual
Provides a complete reference to Slang, the language used to specify a dialogue.

Serpent: C Application Developer's Guide
Serpent: Ada Application Developer's Guide
Describe how the application interacts with Serpent. These guides describe the runtime interface library, which includes routines that manage such functions as timing, notification of actions, and identification of specific instances of the data.

Serpent: Guide to Adding Toolkits
Describes how to add user interface toolkits, such as various Xt-based widget sets, to Serpent or to an existing Serpent application. Currently, Serpent includes bindings to the Athena Widget Set and the Motif Widget Set.

The following figure shows Serpent documentation in relation to the Serpent system:

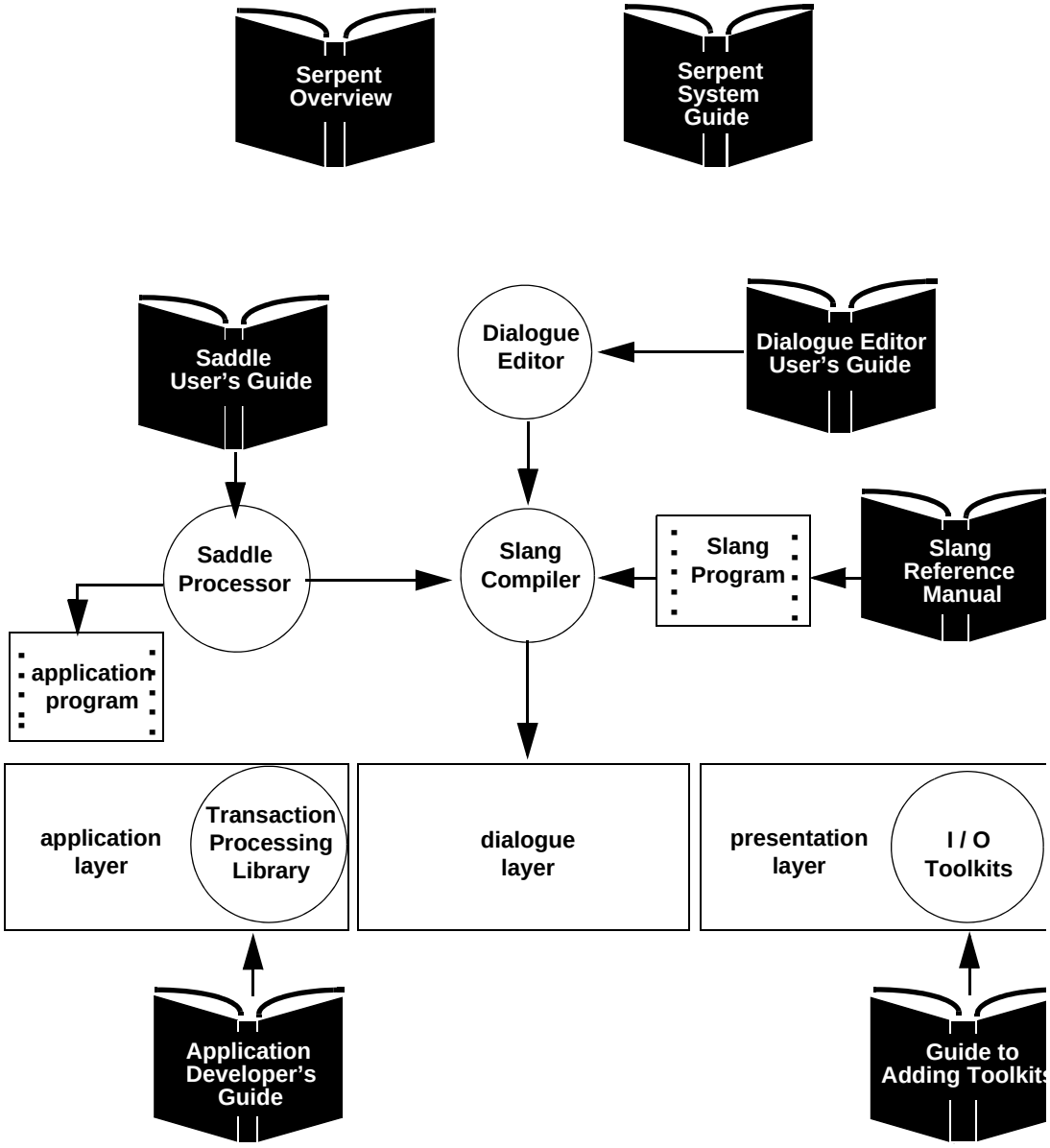

Figure 1-1 Serpent Documents

1.3 The Dialogue Editor

The Serpent dialogue editor is designed to aid the process of creating dialogues and to support an incremental development model of user interfaces. The editor supports the rapid prototyping and structured, top-down development of user interface designs by means of two integrated editors:

1. A layout editor that provides a means of visualizing and creating static prototypes of user interfaces.
2. A structure editor that provides a syntax-directed editor for Slang.

1.3.1 Layout Editor

The layout editor allows you to create a static prototype with the interaction objects (widgets) from various toolkits that have been integrated into the tool. These toolkits currently include both the Open Software Foundation (OSF) Motif and Massachusetts Institute of Technology (MIT) Athena widgets. With the layout editor, a designer can quickly define the layout of a user interface and observe some of the low-level behavior of the widgets. With the editor, the designer can also create instances of interaction objects and specify the attributes of these objects. The attributes specified in the layout editor, unlike those specified in the structure editor portion of the Serpent editor, must have constant values.

1.3.2 Structure Editor

The structure editor provides an interactive approach for specifying Slang dialogues. It provides a series of forms and menus for defining the dynamic and presentation portions of the user interface. The structure editor supports the specification of any user interface that can be created using Slang.

1.4 Using the Mouse

The dialogue editor requires the use of a three-button mouse. These buttons are referred to as the “left,” “middle,” and “right” buttons throughout the remainder of this document. Other terms required for menu selections and moving and resizing objects in the dialogue editor are defined in Section 2.1.6, and terms used to manipulate the objects under the control of a window manager are listed in Section 2.2.2.

The following items describe some basic actions you can take with the mouse:

select object
To select an object, move the mouse cursor inside the object and "click" on the left mouse button. Clicking on an object indicates the object that subsequent dialogue editor commands will affect.

pick object
To pick an object, move the mouse cursor inside the object and, holding down the shift key, "click" on the left mouse button. This interactive technique is often used for identifying objects as parameters to the various dialogue editor commands.

drag object
To move an object across the screen without changing its parenting relation, move the mouse cursor over an object and press a mouse button. If this object can be dragged, the mouse button will attach itself to the object so that the object follows the mouse, within its constraints.

scrolling
To move vertically through the file, use the rectangular area containing a slide region and a slide bar. When the center mouse button is clicked the slide bar moves to the current pointer location. The slide bar can also be dragged using the center mouse button.

move object
To move an object and simultaneously change its parenting relation, move the mouse cursor inside the object and, simultaneously holding down the shift key and the right mouse button, move the mouse slightly until the mouse cursor becomes a four headed arrow  . When you have positioned the object, release the right mouse button.

resize object
To change the size of an object, move the mouse cursor inside the object and, holding down the shift key and the middle mouse button, position the mouse button in the lower right-hand corner of the object. Drag the corner outward to resize the object.

text entry
To enter or modify text, move the mouse cursor into a text entry field. When the mouse cursor changes from an arrow head to an **`I`** beam, you can enter or modify text in that field. To save the text you have entered or modified in that field, press the **`Return`** key. (The text will not be modified unless **`Return`** is pressed.)

selecting menu items
To select an item on a menu, move the mouse cursor to the desired item and press the left mouse button. Release the mouse button to select the item.

2 Getting Started

This section presents a simple example of how to develop user interfaces with the dialogue editor. The example consists of constructing a counter that has a simple user interface with limited dynamic interaction.

The counter program (in demos/saw/counter directory) allows a user to increment a counter repeatedly by selecting a button labeled **Push Me**. When the user selects a button labeled Q**uit**, the program terminates. The counter program consists of a label widget (`xawlabel3` in Figure 2-1) displaying the value of the counter, a command widget used to increment the counter, a command widget to terminate the program, and a form widget that provides a background for this example. The display for the counter program is illustrated in Figure 2-1.

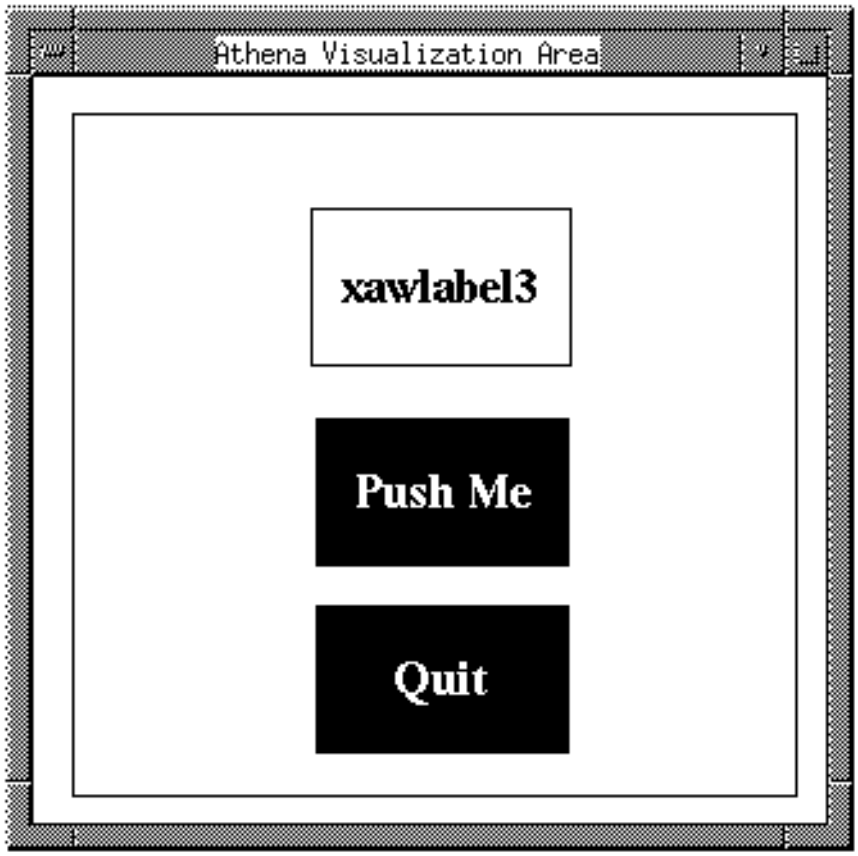

Figure 2-1 Counter Display

The counter program contained in the demos/saw/counter directory is very similar to this example. You may wish to run it to familiarize yourself with the user interface and the behavior of the counter program before reading the remainder of this chapter.

2.1 Defining Layout and Presentation

This section presents the steps necessary to lay out and define the presentation for the counter example. You will be asked to use the layout editor to create several interaction objects such as a background form, command widgets, and a label widget. You will then specify variables and methods associated with the interaction objects. After generating a Slang file, you will compile and run the sample program. Although this counter example uses the Athena widget set, you can use the Motif widget set for the example as well.

The steps are:

1. Start the dialogue editor (Section 2.1.1).
2. Load a new dialogue (Section 2.1.2).
3. Start the layout editor (Section 2.1.3).
4. Create the form widget (Section 2.1.5).
5. Resize the form widget (Section 2.1.6).
6. Parent the command widget that increments the counter to the background form so that they are subsequently treated as one object (Section 2.1.7).
7. Change the attribute values of the objects (Section 2.1.7).
8. Change the command widget label to **Push Me** (Section 2.1.9).
9. Invert the foreground and background colors (Section 2.1.10).
10. Change the font to Times, regular slant, bold face text with a point size of 18 (Section 2.1.11).
11. Duplicate the command widget and change the label of the new command widget to **Quit (S**ection 2.1.12).
12. Create a label widget; position, resize, and change the font. (Section 2.1.13).
13. Save and execute the dialogue (Section 2.3)

2.1.1 Changing the Workspace

The dialogue editor uses three main windows: the main workspace, the toolbox, and the layout area. These windows are managed by the window manager installed on your system. They are moved and resized using normal window manager commands for these operations.

2.1.2 Starting and Quitting the Dialogue Editor

To start the dialogue editor, type `ded` at the Unix prompt. The current directory becomes the working directory: by default, any files that are saved or loaded from the editor will reside in the current directory. The `ded` command displays the menu bar and the main workspace for the dialogue editor. This main workspace window initially takes up the entire screen but may be resized and/or moved using the appropriate controls in your window manager.

To quit the editor at any time, pull down the **System** menu and select the `Quit` menu item, as illustrated in Figure 2-2.

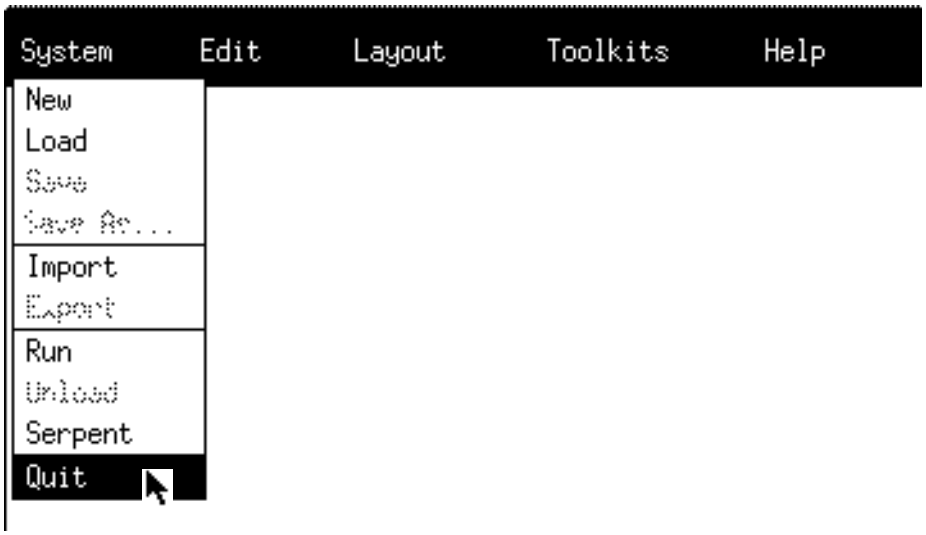

Figure 2-2 Exiting from `ded`

2.1.3 Loading a New Dialogue

To create a new dialogue, specify the `ded` command and then select the `New` command from the **System** menu. If you don't specify a parameter with the `ded` command, the dialogue editor comes up without loading a dialogue. This command displays the form illustrated in Figure 2-3.

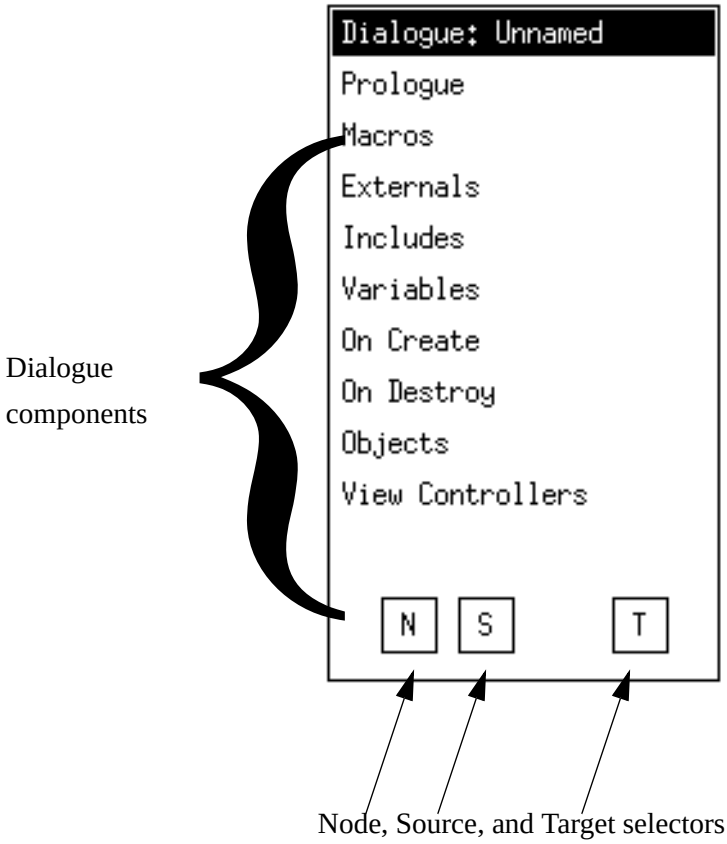

Figure 2-3 Dialogue Construct

The dialogue construct form consists of a title bar showing the dialogue name, a menu of dialogue components, and buttons (`node`, `source`, and `target`) used to identify the dialogue as a parameter in many of the commands supported by the editor.

This form is used later in this example to specify the dynamic portion of the counter example.

2.1.4 Displaying the Layout Area

Once a dialogue has been loaded, you can begin by displaying a layout area for either the Athena or Motif widget set by selecting the `Athena layout` or `Motif layout` command from the **Layout** menu. Selecting `Athena layout` displays both the layout area for Athena widgets (Figure 2-4) and the Athena toolbox containing icons for the Athena toolkit (Figure 2-5). The same procedure is used for the Motif widget set. For the rest of this example, the Athena widget set will be used.

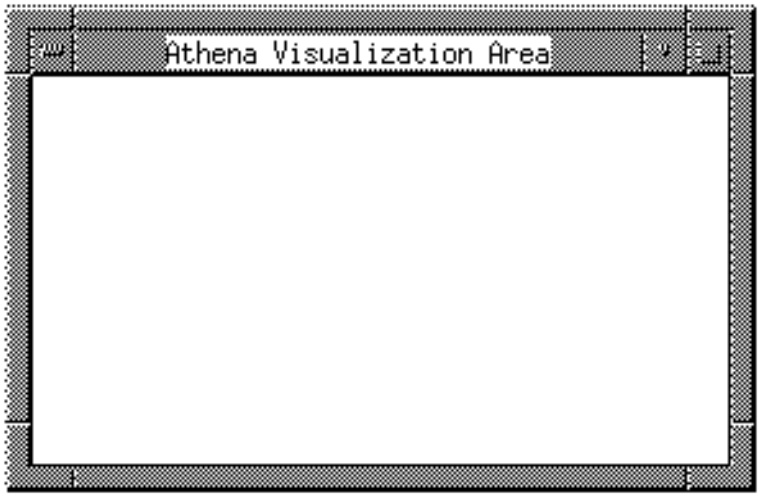

Figure 2-4 Athena Layout Area

Note: There will be a total of three widgets required in the work area in the example displayed in Figure 2-4, so position and size the command widget accordingly.

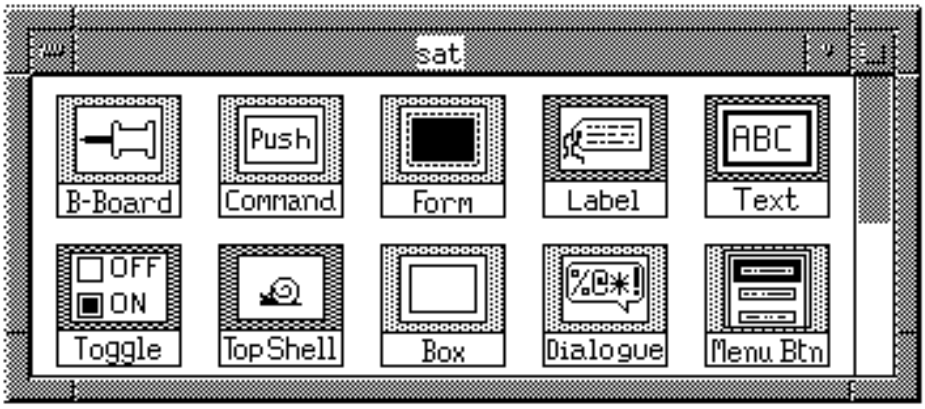

Figure 2-5 Athena Toolbox

The Athena toolbox provides access to the interaction Athena objects that have been integrated into Serpent. Use your window manager to manipulate the toolbox area.

Note: If you are using Motif window manager, as you start to move the window, the mouse cursor changes from a single arrow to a four-headed arrow ✥, indicating that you may move the window.

2.1.5 Creating a Background Form

The next step in creating the counter example is to create the background form. Select the **Form** icon from the Athena toolbox window using the left mouse button. The icon is displayed in reverse video to indicate it has been selected, as illustrated below:

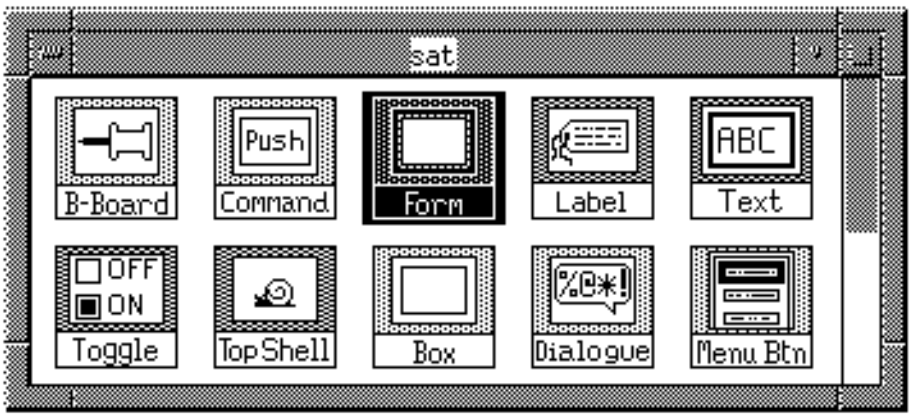

Figure 2-6 Athena Toolbox with Form Icon Selected

After the icon has been selected, the status area in the main system menu will also display the following message:

```
Status: Select position in visualization area to draw menu.
```

To position the object, move the mouse pointer into the Athena layout area and click the left button. The upper left-hand corner of the object will be positioned at the x, y location indicated by the mouse pointer. After the object has been created, the following message is displayed in the status area:

```
Status: xawform0 created.
```

2.1.6 Resizing and Moving Widgets

To enlarge the **Form** icon so the other widgets can be placed inside, hold down the shift key and the middle mouse button and drag the right bottom corner as shown in Figure 2-7.

.

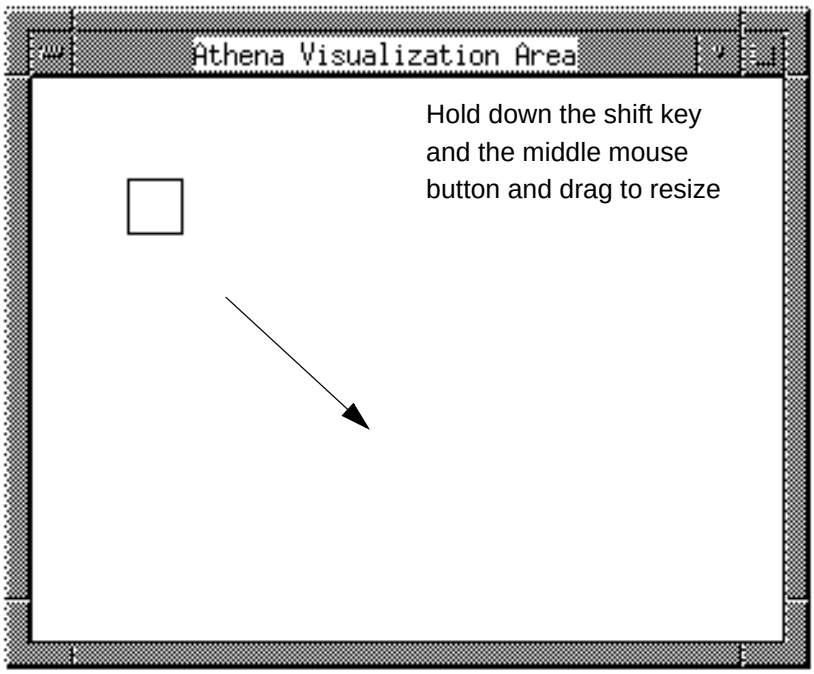

Figure 2-7 Form Widget

After you have finished resizing the form, the following message should appear in the status area:

```
Status: xawform0 resized.
```

Your display should appear similar to the one in Figure 2-8.

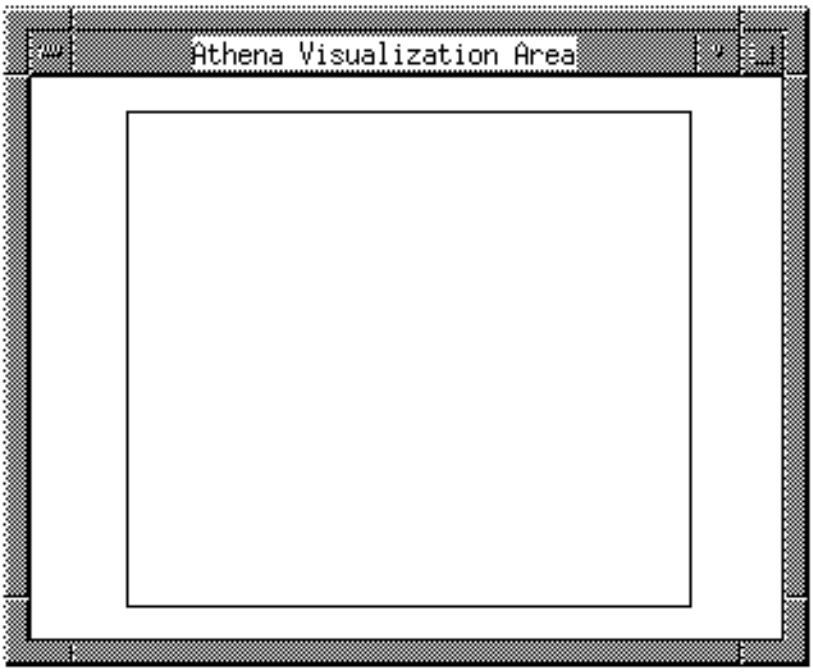

Figure 2-8 Enlarged Form Widget

To move the form, press the **Shift** key and the center mouse button and drag the form. Widgets may also be moved by holding down the shift key and the right mouse button. Moving an Athena widget causes the x, y, horizontal, and vertical distance parameters to be updated. Moves are also reported in the status area.

2.1.7 Creating an Object With a Parent

The next step is to add the command widget that increments the counter. Select the **Command** icon from the Athena toolbox and select a position within the form widget already in the layout area. Selecting the **Command** icon causes the new command widget to be directly parented to the form; that is, the widget is attached to the form so that the widget and the form will be treated as a single object in future operations, as displayed in Figure 2-9. The label field of the widget is initialized to the name of the object instance.

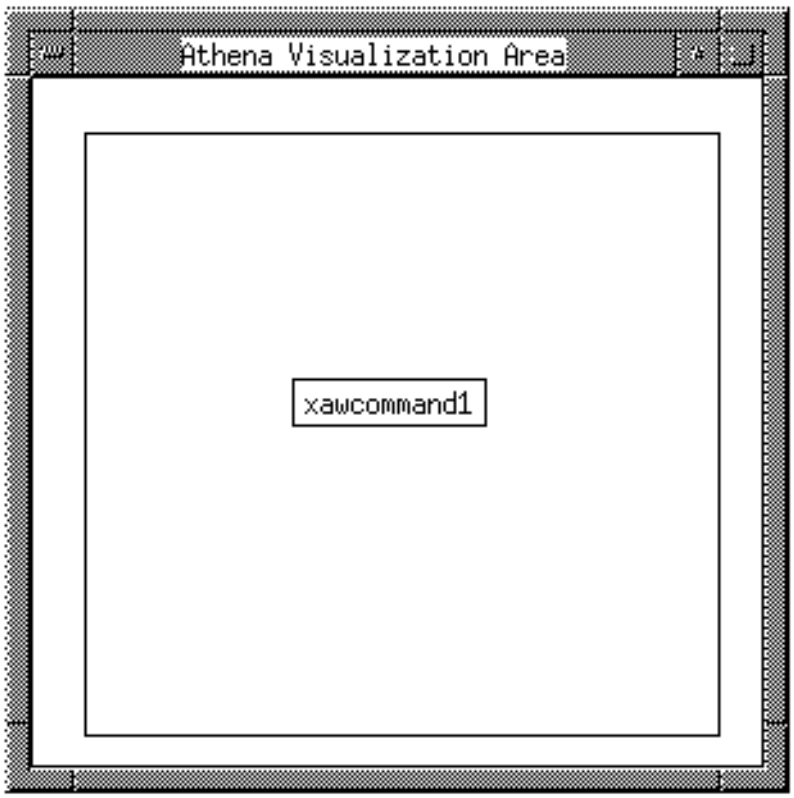

Figure 2-9 Command Widget Inside Form Widget

The command widget can be moved or resized as described in Section 2.1.6.

2.1.8 Changing Attribute Values

To review, attributes of position and size can be modified through the layout editor using direct manipulation techniques. To change other attributes of the objects, you need to bring up the edit attributes form for a particular object. To do this, first select the icon using the left mouse button. A message should appear in the status area indicating that the icon has been selected. Selecting the `Edit Attributes` command from the **Edit** menu will display the actual form. Newly created objects are automatically selected by the editor. The edit attributes form for the Athena command widget is illustrated in Figure 2-10.

Note: If you pull down the Edit menu and the selections are grayed out (not selectable), it means no object has been selected or an edit attribute form is already displayed. If you exit the menu and select the command widget, the menu selections will become available.

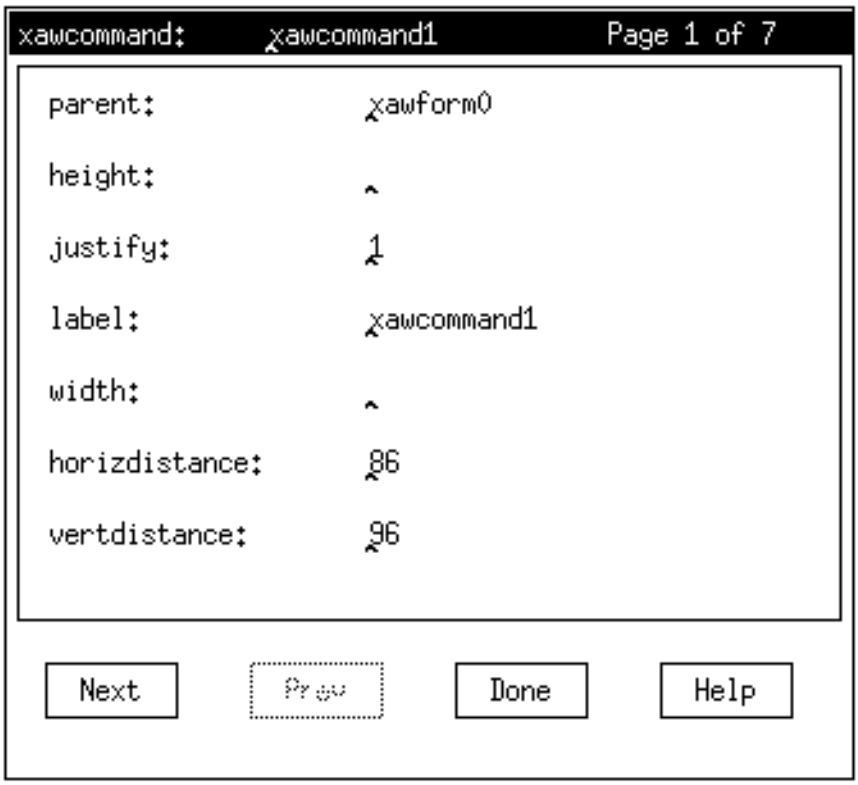

Figure 2-10 Edit Attributes Form

The form containing edit attributes can contain more than one page per screen, depending on the number of attributes defined for each object type. The **Next** and **Prev** buttons can be used to move through the different pages. The title bar contains the type and name of the object associated with the form.

Many of the attributes contain existing or default values. Default values are not explicitly set, so they may be changed globally through the Glue files. Pressing **Return** for any given attribute will explicitly set the value; even if the default is changed the attribute will still have this explicitly set value. See the *Serpent: System Guide* (CMU/SEI-91-UG-2) and *Serpent: Guide to Adding Toolkits* (CMU/SEI-91-UG-8) for more information on changing default values through the Glue files.

Each attribute has a specific type, and each attribute type has an attribute “editor” particular to the attribute type. The following subsections will demonstrate how to modify several attribute types in the context of the counter example.

2.1.9 Changing String Attributes

The first attribute of the command widget that needs to be changed is the label attribute. To change the attribute, position the mouse pointer over the current text for the attribute and type `Push Me`. Pressing **Return** causes the new value to be applied to the command object in the Athena layout area.

A fairly extensive set of *Emacs*-style editing commands are available for editing within the text area. The following are some of the *Emacs* commands available:

Backspace: delete previous character

Control-D: delete current character

Control-K: delete the rest of the line at cursor position

Arrow keys: move the cursor one character at a time in the direction of the arrow

2.1.10 Changing Color Attributes

Another attribute type is color. Each color attribute has a push button labeled with the letter C on the right-hand side of the attribute dialogue box, as illustrated in Figure 2-11. Selecting this button will bring up the color palette form shown in Figure 2-12. Colors may be modified through this palette or by typing the name of the color at the prompt.

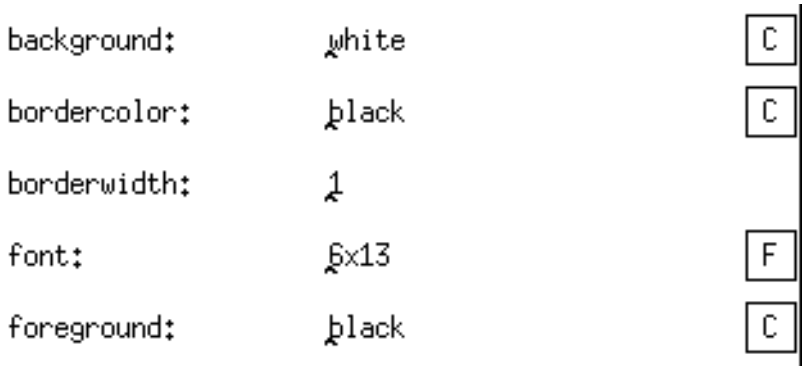

Figure 2-11 Color and Font Attributes

To demonstrate the process by which a color attribute can be modified, foreground and background colors for the command widget are inverted in this manual. If you are using a color display you may use any colors you would like in this example. To change the background color, select the push button on the background attribute line to bring up the color palette. Now select a background color. On a monochrome display, all the colors are mapped to either black or white. If you are using a monochrome display, you may just want to change the background color to black.

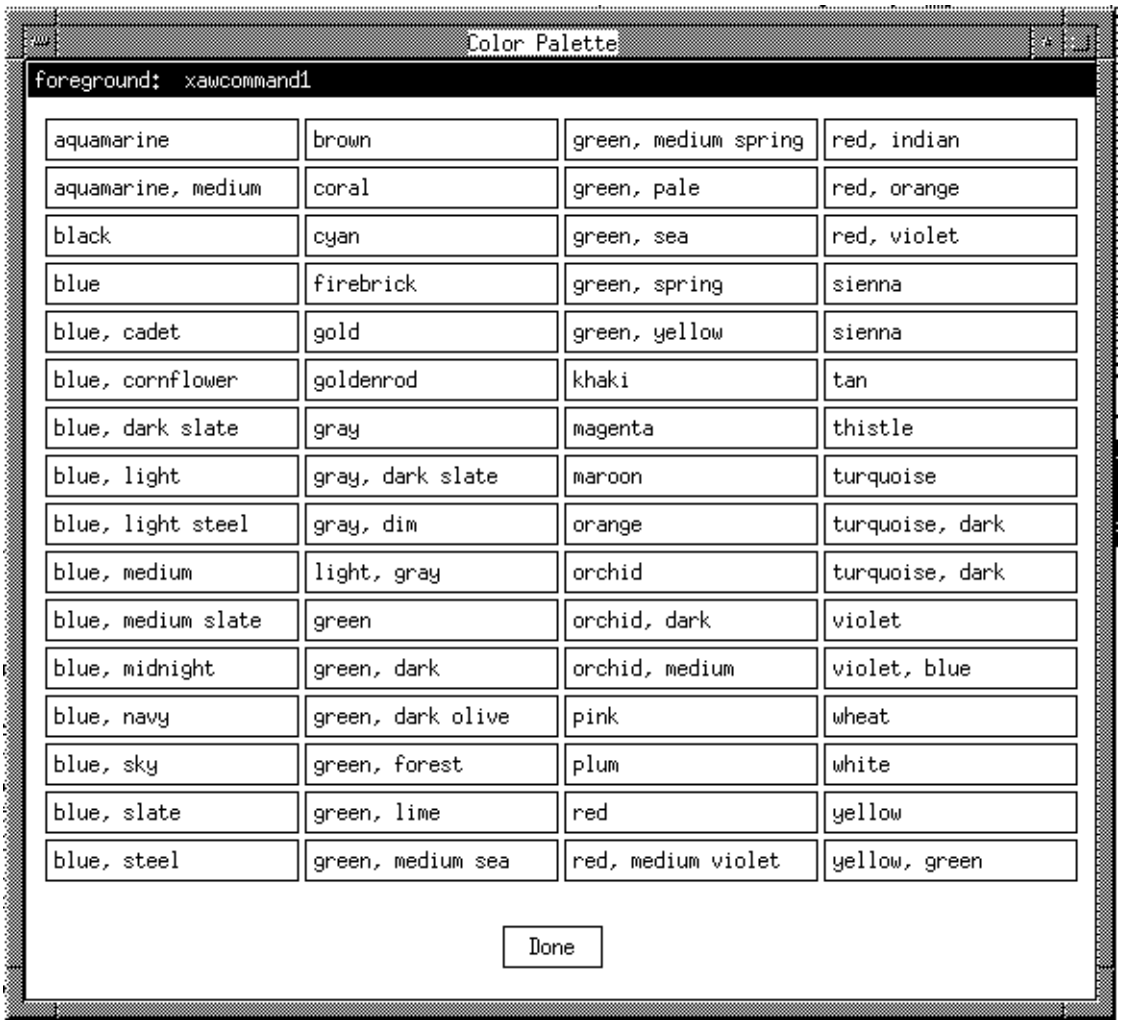

Figure 2-12 Color Palette

After the selection has been made, two things happen. First, the background color of the command widget changes from white to whatever color you selected. Second, the name of the color appears in the text area of the attribute. If you know the name of the color you want, you can change the color by entering the name directly into this text field.

If you have not yet pressed the **Done** button on the color palette form, you can simply press the **C** push button on the foreground attribute line. This allows you to use the color palette to change the foreground color. If you are inverting the color of the object, you may want to change the foreground to white. Since the color palette is a separate window, you can simply move it out of the way or iconify it for later use. This will save you some time in redisplaying the form.

The version of the color palette displayed in Figure 2-12 is displayed on monochrome displays. A separate, color version of the color palette is displayed on color displays.

2.1.11 Changing Fonts

The font attribute for the command widget is on the same page as the foreground and background color attributes. Font attributes are modified in similar fashion to color attributes. If you know the name of the font, you can enter it directly in the text area; otherwise, select the **F** button to bring up a font selector dialogue box. For this example, choose Times for the font type, r for regular slant, bold face text, and a point size of 18. As you select different font styles and attributes, the font selector shows which fonts fitting that description are available by graying out the unavailable choices. When you have finished specifying this font, the font selector should appear as in Figure 2-13. Now select the **Apply** button so that the new attribute information will be displayed in the command widget.

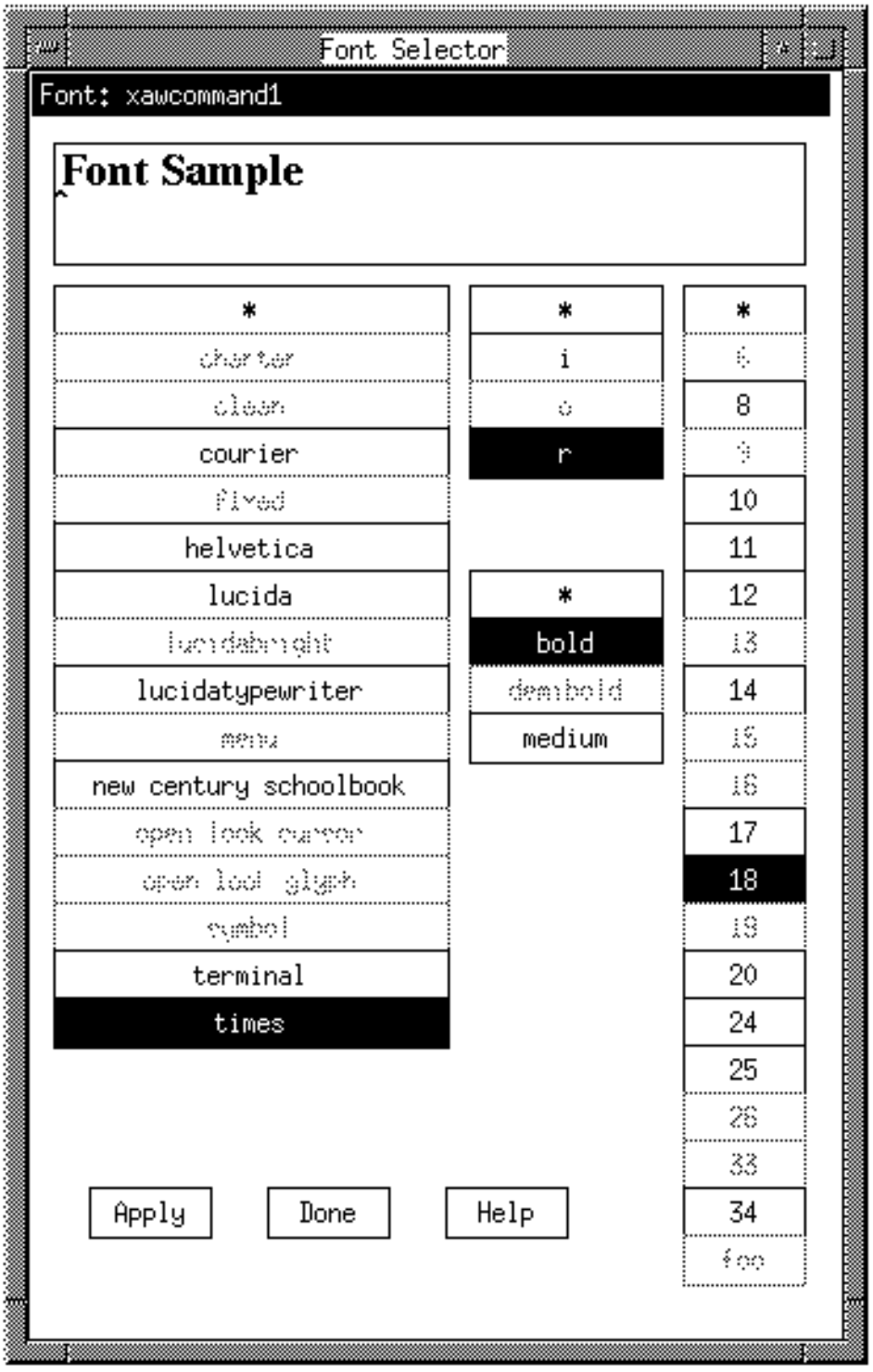

Figure 2-13 Font Selector

The area containing the text `Font Sample` is a text widget that allows you to preview your font selection before applying it to the actual object. The text in this area can be edited using the *Emacs* editing commands previously described so that the text more closely resembles the target application.

Like the color palette, the font selector can also be left on the screen and used to modify other font attributes in this or other objects. To do this, select the **F** button to the left of the attribute. The font selector is also a separate window that can be moved or iconified if it is using too much display space.

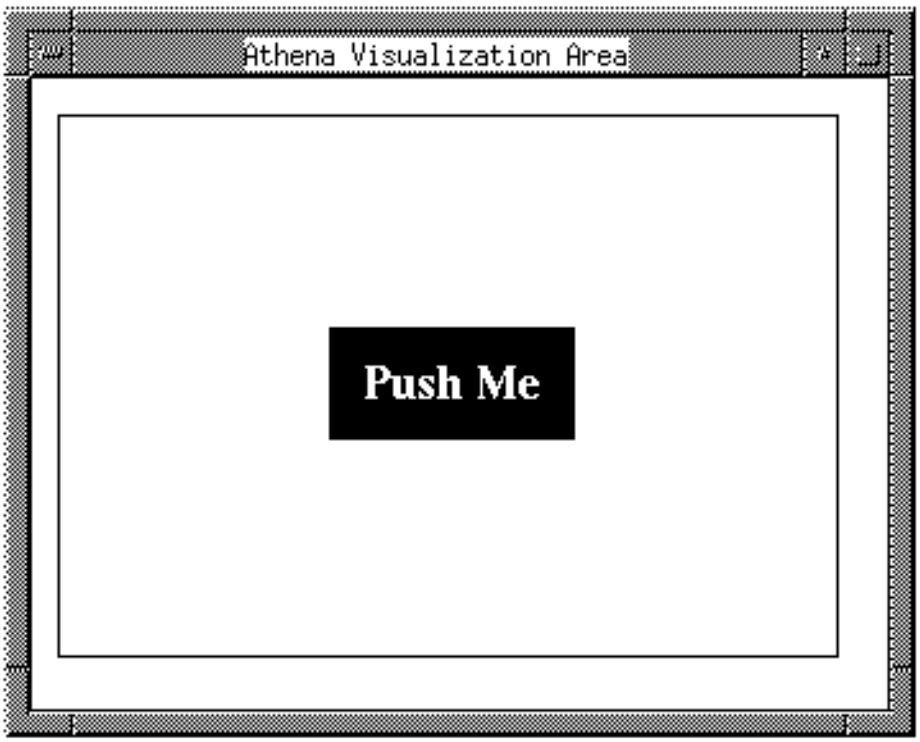

Figure 2-14 Command Widget with Modified Attributes

Figure 2-14 illustrates the current status of the Athena visualization area after the command widget attributes have been modified. At this point, you should exit from the edit attributes form by selecting the **Done** button.

2.1.12 Duplicating an Object

The next step in creating the counter example is to create a **Quit** button. Since this button has the same object type as the label widget, you can simply duplicate the **Push Me** button to create the basic template for the **Quit** button. Choose the **Push Me** button by holding down the shift key and clicking on the button with the left mouse button. The following message should appear in the status area:

```
Status: xawcommand1 selected.
```

Once you have selected the **Push Me** button, pull down the **Edit** menu and select the `Duplicate` command. The dialogue editor creates a copy of the command widget with all modified attributes. Figure 2-15 displays the results of the `Duplicate` command. The position is offset by 25 to make the new object visible on the screen.

Figure 2-15 Results of Duplicate Command

Move the new command widget down and to the left by holding down the **Shift** key and the right mouse button and dragging the object to the new position.

Note: Alignment of the widgets can be simplified by selecting `Grid Snap On` from the **Visualize** pull-down menu.

The next step is to replace the label string with the correct text. To do this, select the new command widget, followed by the `Duplicate c`ommand from the **Edit** menu. Modify the text using the *Emac*s commands specified in Section 2.1.9. Press the **Return** key with the mouse pointer in the label field, making sure that the display in the Athena layout area is updated. Once you have completed this change, select the **Done** button on the edit attributes form to remove it from the display.

2.1.13 Completing the Layout

Since a label widget differs from a command widget, it is inappropriate to duplicate any of the existing command and form widgets. To create a label widget parented to the background form, follow the instructions in 2.1.7. Resize and move the label to the appropriate location and then match the font to the one used in the command widgets `(-*-Times-bold-r-*-*-1)`. If you have not selected the **Done** button on the font selector form, bring up the edit attributes form for the object, select the **F** button to the left of the font attribute to associate the font editor with the attribute, and select the **Apply** button in the font selector form.

You have now specified the static presentation portion of the user interface. The interface should closely resemble the original counter display illustrated in Figure 2-1 for the counter example. The following section deals with adding dynamic interactions to the counter example.

2.2 Adding Dynamic Interactions

This section describes the steps necessary to define the dynamic interactions for the counter example:

1. Define a counter variable (Section 2.2.1).
2. Constrain the label for the **Label** widget to the value of counter (Section 2.2.2).
3. Define a method for the **Push Me** command widget that increments the value of the counter when the button is selected (Section 2.2.3).
4. Define a method for the **Quit** button that causes the counter program to exit (Section 2.2.3).

2.2.1 Creating a Variable

As described in section 2.1.3, the `new` command loaded a new dialogue construct form. This dialogue construct is the top-level construct for the counter dialogue. To create a variable within the dialogue construct, select `Variables` from the list of components defined within the dialogue construct, as shown in Figure 2-16.

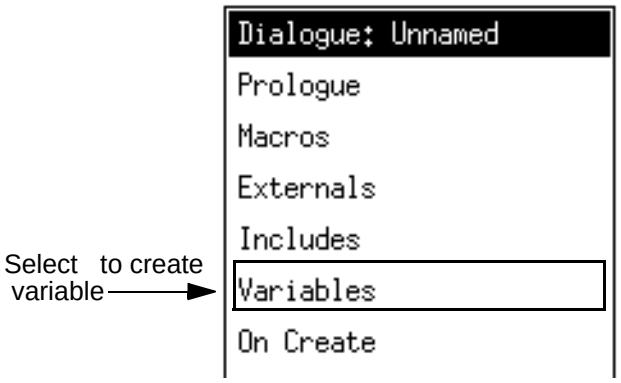

Figure 2-16 Displaying Dialogue Variables

This displays the list of variables already defined for the dialogue (this list should be empty, in this case) and a **New** button. Selecting the **New** button causes a new, unnamed variable to be created, as shown in Figure 2-17.

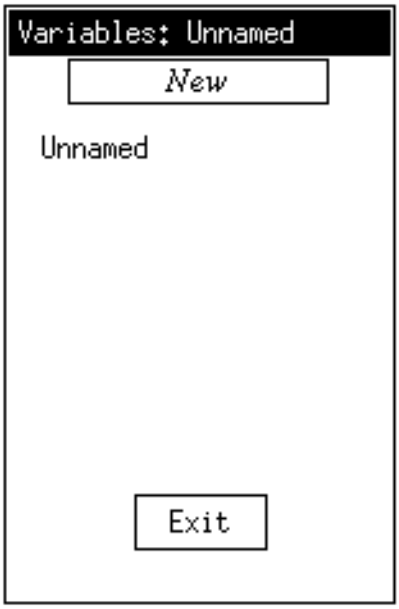

Figure 2-17 Variable List

Select `Unnamed` to edit the variable. A separate form is displayed that contains a title bar, a text entry area, and several buttons, as shown in Figure 2-18. To change the name of the variable, position the mouse pointer within the text field containing the current name (`Unnamed`) and enter the new name of the object. Again, standard *Emacs* editing commands can be used. Once you have entered the new name, press the **Return** or **Enter** key before moving the cursor. The name of the variable is updated in the list of variables.

Next, move the mouse pointer into the large text area and enter the initial value for the variable, in this case 0. (This may be entered with or without the trailing ";" separator character.) Since this is a multiple-line text area, pressing the **Return** key simply moves the cursor to the next line. To save the initial value, you must select the **OK** button at the bottom of the display. You could also select the **Cancel** button, which removes the form from the display without changing the value of the variable.

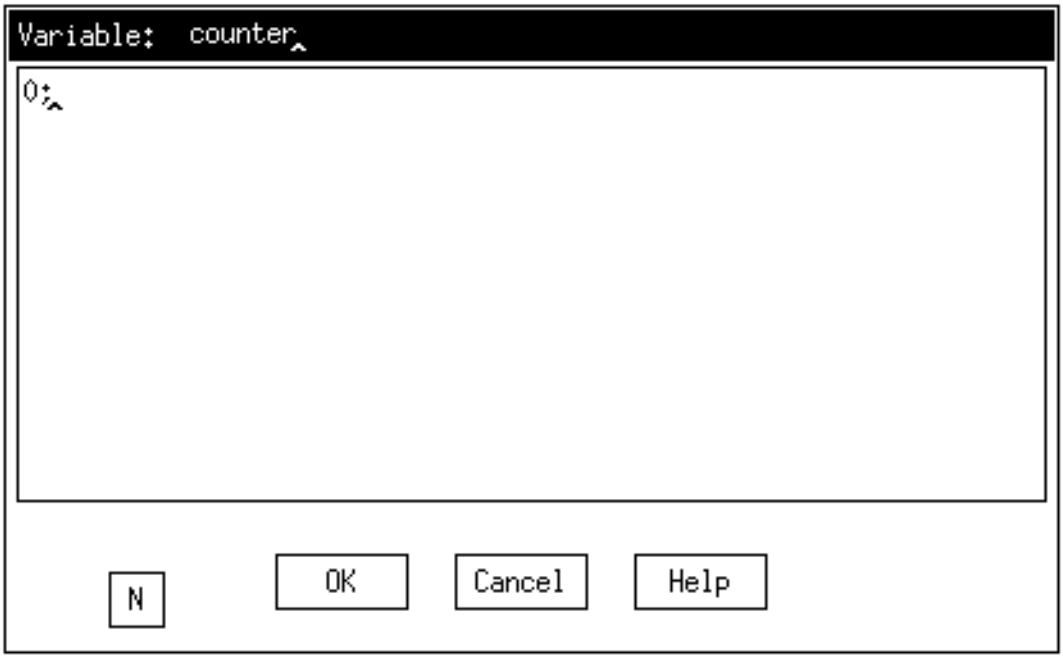

Figure 2-18 Variable Definition

2.2.2 Specifying Attribute Constraints

Until now, you have been creating object instances and assigning constant values to the object instance attributes. For each object instance created in the layout area, a separate object template has been created in the dialogue. These object templates can be accessed through the dialogue construct form by selecting the `Objects` menu item. Figure 2-19 illustrates the object list form. Each of the object instances created in the layout area should be listed here as well.

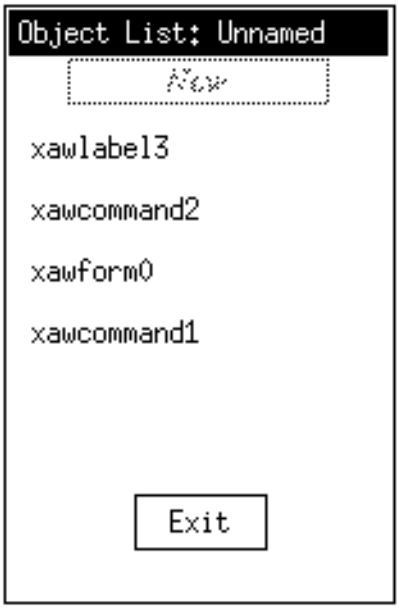

Figure 2-19 Object Template List

To link the label attribute value of the `xawlabel3` object to the value of the `counter` variable, first select the object to bring up the object form. The object template form for the `xawlabel3` object is shown in Figure 2-20.

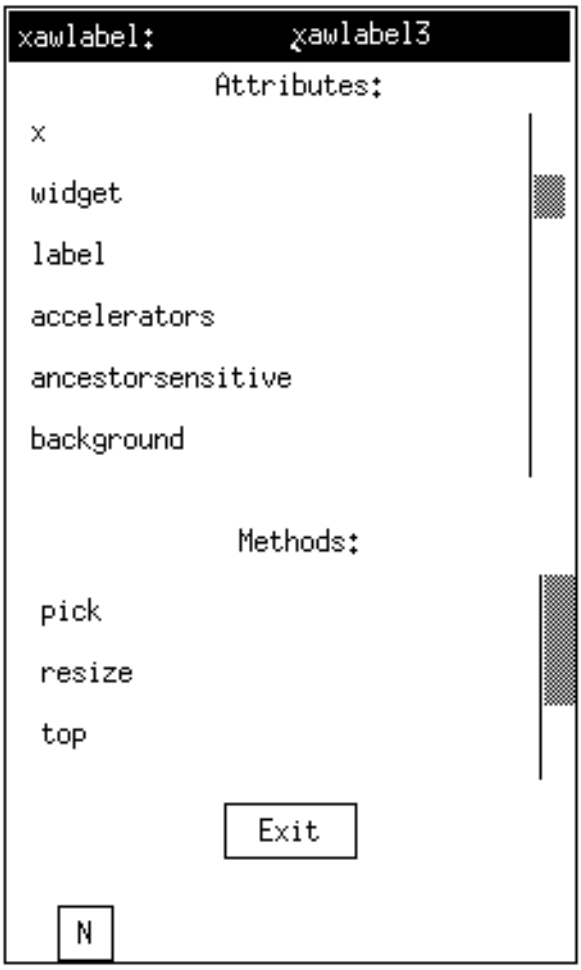

Figure 2-20 Object Template Form

It may be necessary to scroll down to find the label widget. Once you have located the **Label** attribute, select it to bring up the attribute form. The attribute form (shown in Figure 2-21) consists of a title bar containing the name of the object template and attribute being edited, a text area used to display/edit the attribute definition, and the **OK**, **Cancel,** and **Help** push buttons along the bottom of the form.

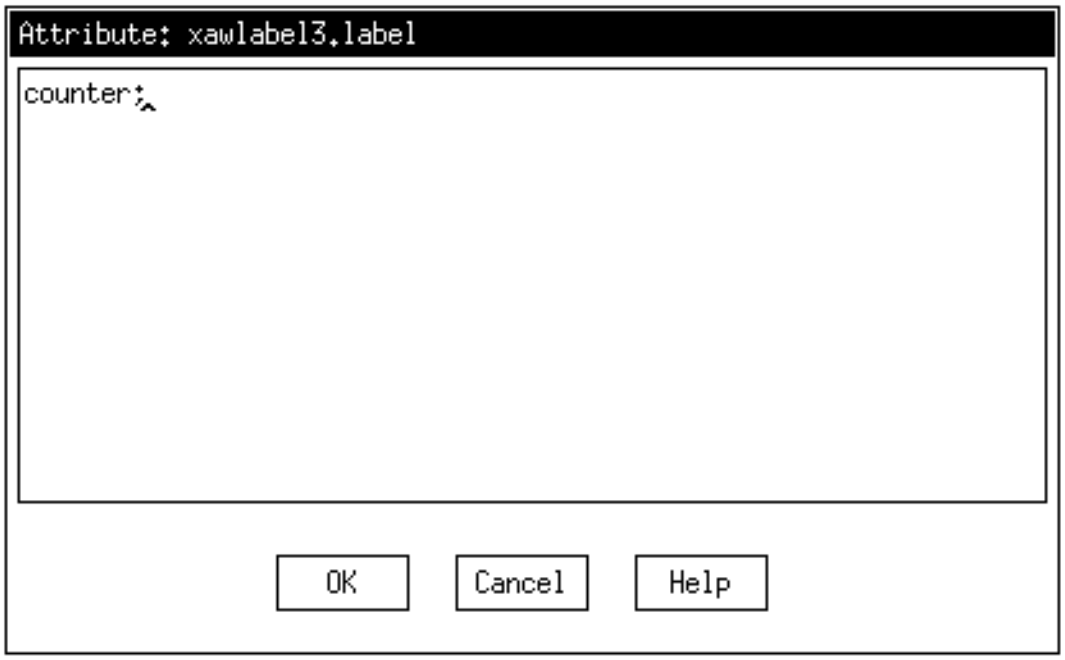

Figure 2-21 Attribute Form

To replace the current definition of the label attribute, position the cursor over the text area, enter the variable name `counter`, followed by a semi-colon, and select the **OK** push button. This links the value of the label to the value of the counter in the counter dialogue.

2.2.3 Specifying a Method

The next step in creating the counter dialogue is to specify the methods associated with the P**ush Me** and **Quit** command widgets. Returning to the object template form shown in Figure 2-20, select `xawcommand1` (the name of the **Push Me** command widget) from the list of objects. You can find the name of an object instance by picking the object in the layout area. The name of the object is then displayed in the status area along with an indication that the object has been selected.

Find and select the `notify` method associated with the command widget. This displays the method form shown in Figure 2-22. Enter the text shown to increment the counter and select the **OK** button. This causes the method definition to be updated and the form to be removed from the display.

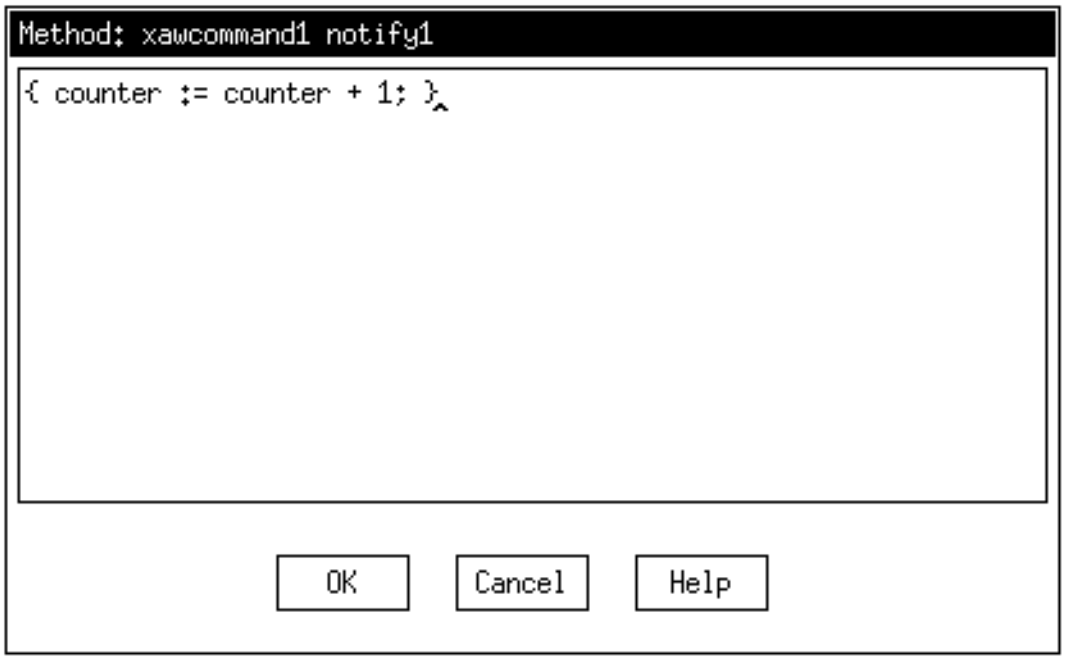

Figure 2-22 Method Form

Repeat this process for the **Quit** command widget, `xawcommand2`. The method definition is:

```
{exit();}
```

Once you have completed this step, you have completely specified both the static and dynamic portions of the counter user interface dialogue and you are ready to save and test your work. Saving and executing the counter program is described in the next section.

2.3 Saving and Executing a Dialogue

This section describes the steps involved in executing a dialogue.

2.3.1 Saving a Dialogue

Once you have completed the specification of a dialogue, you need to save your work to disk. Select the `Save` command from the **System** menu. If the dialogue has already been named, the file is saved directly to the named file. If the dialogue is still unnamed, the dialogue box shown in Figure 2-23 is displayed.

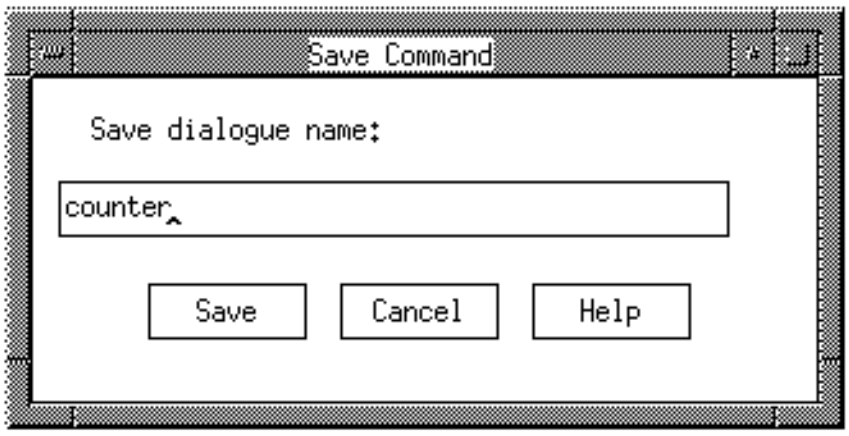

Figure 2-23 Save Dialogue Box

To save the dialogue type in the file name, for example `counter`, select the **Save** button or press the **Return** key. The file name is automatically given a default extension of `.slb`, indicating a slang binary file, since it is more convenient to store dialogues in that format. The `slb` format can be reloaded into the dialogue editor with the `Load` command. Therefore, it is not necessary that you add your own file extension to the file name.

2.3.2 Exporting a Dialogue

To compile and run a dialogue, it is necessary first to generate a Slang representation of it. Select the `Export` command from the **System** menu. The export dialogue box shown in Figure 2-24 is displayed. The default name for the generated Slang representation is the name of the dialogue. The `.sl` Slang extension is automatically appended to the end of the file name. Once this file has been generated, you are ready to compile and run the counter program.

Figure 2-24 Export Dialogue Box

2.3.3 Compiling a Dialogue

To compile, link, and run the counter program select the `Run` menu entry from the **System** menu. This causes the run dialogue box to be displayed, as shown in Figure 2-25. Enter the name of the dialogue (the `.sl` extension is assumed) and select the **Run** push button.

Figure 2-25 Run Dialogue Box

The dialogue editor will compile, link, and execute the Slang program called counter. The counter program will come up as a separate process in a separate window.

2.3.4 Final Counter

The following output is the final description in the Slang code of the counter example as generated by the editor. The code has been edited to increase its readability:

```
#include "sat.ill"

|||

VARIABLES:
 counter: 0;

OBJECTS:
 xawform0: xawform {
    ATTRIBUTES:
      parent: NULL;
      height: 242;
      width: 262;
      x: 15;
      y: 22;
      horizdistance: 15;
      vertdistance: 22;
 }

 xawcommand1: xawcommand {
    ATTRIBUTES:
```

```
    parent: xawform0;
    background: "black";
    font: "-*-times-bold-r-*-*-18-*-*-*-*-*-*-*";
    foreground: "white";
    height: 54;
    label: "Push Me";
    width: 103;
    x: 79;
    y: 95;
    horizdistance: 79;
    vertdistance: 95;
  METHODS:
    notify:
      { counter := counter + 1; }
}

xawcommand2: xawcommand {
  ATTRIBUTES:
    parent: xawform0;
    background: "black";
    font: "-*-times-bold-r-*-*-18-*-*-*-*-*-*-*";
    foreground: "white";
    height: 54;
    label: "Quit";
    width: 103;
    x: 79;
    y: 169;
    horizdistance: 79;
    vertdistance: 169;
  METHODS:
    notify:
      { exit(); }
}

xawlabel3: xawlabel {
  ATTRIBUTES:
    parent: xawform0;
    font: "-*-times-bold-r-*-*-18-*-*-*-*-*-*-*";
    height: 65;
    label: counter;
    width: 101;
    x: 80;
    y: 18;
    horizdistance: 80;
    vertdistance: 18;
}
```

Example 2-1 The Counter Example

3 Menubar Example

This section presents a sample program that demonstrates how to use the dialogue editor to create a pull-down menu user interface. You can follow the example, creating objects similar to those in the counter example, such as a background form and a group of command widgets. You will also create a group of widgets and move them into a view controller.

3.1 Creating the Menubar

The menubar program demonstrates the creation of a user interface for a pull-down menu. The user interface consists of a menubar with 3 selections: **Menu1**, **Menu2** and **Quit.** When a user selects either **Menu1** or **Menu2**, another pull-down menu is displayed. When items in the pull-down menu are selected, they reverse foreground and background color. When **Close Menu** is selected, the pull-down menu closes. The final layout of the widgets should look like Figure 3-1.

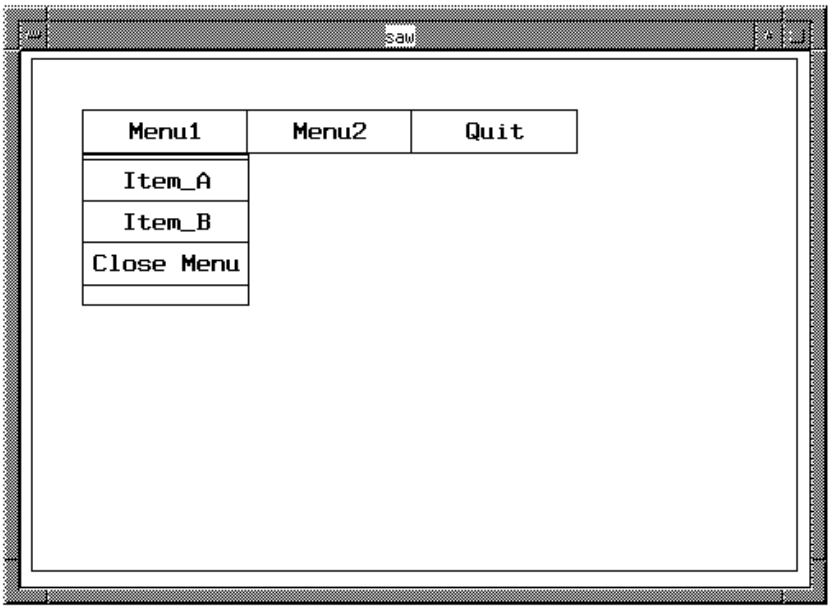

Figure 3-1 Pull-Down Menu Example

3.1.1 Creating a Workspace and a Form Widget

Follow the instructions outlined in Section 2.1 through Section 2.1.5 to create a workspace similar to that for the counter program. When you complete these steps, you should have a workspace with a resized form widget, as shown in Figure 3-2.

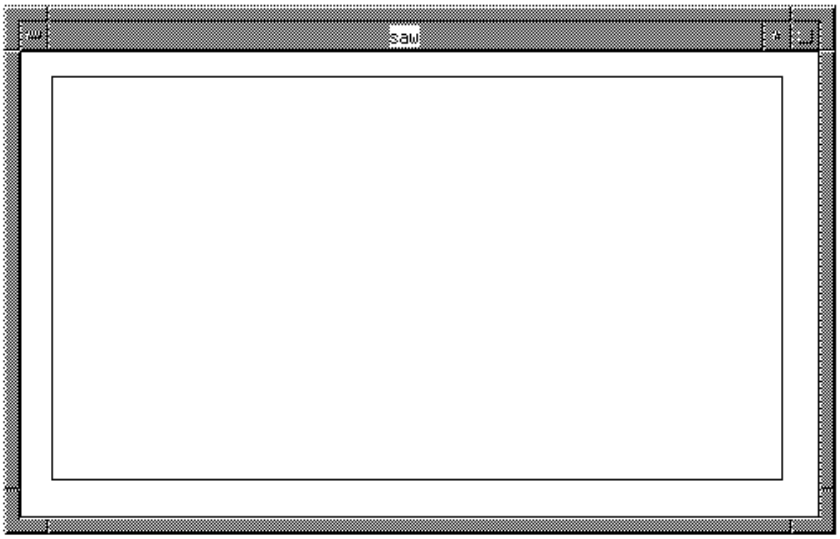

Figure 3-2 Menubar Workspace

3.1.2 Creating a Command Widget

Next, create a command widget as you did in Section 2.1.4.

Note: Before you start, select the `Visualize` entry from the menubar and turn on `Grid Snap`**.** This will make it easier to align the command widgets.

Once you have created the command widget, select `Edit Attributes` from the **Visualize** menu. Then change the size of the command widget to the following: Height: 25, Width: 100.

Next change the label text from “command” to “Menu1.” Now select page 2 of the attribute form and select the **F** box to bring up the font selector. Choose the following font: fixed for the font name, r for regular slant, bold face text and a point size of 15. Make sure you select the **Apply** button to display and save the font in the command widget. Your workspace should now look like that shown in Figure 3-3.

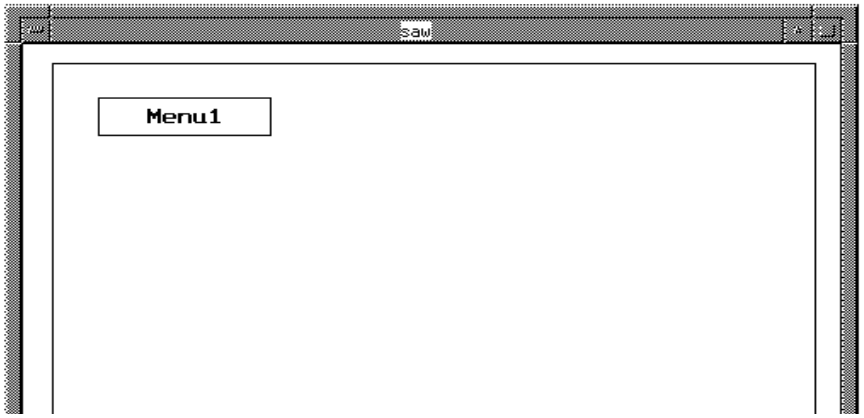

Figure 3-3 Menu1 Command Widget

3.1.3 Duplicating the Menu1 Command Widget

Select the **Menu1** command widget, then select the `Duplicate` menu entry from the **Edit** menu. Move the new command widget so that it is just to the right of the **Menu1** widget. Select `Edit Attributes` from the **Visualize** menu and change the label text to "Menu2 . "

Note: A quick way to change the text is to move the cursor into the text field. Once it has changed to the I beam, enter Control-E. This will place the cursor at the end of the line. Then backspace to erase the "1," replace it with a "2," and press Return.

Your workspace should look like Figure 3-4.

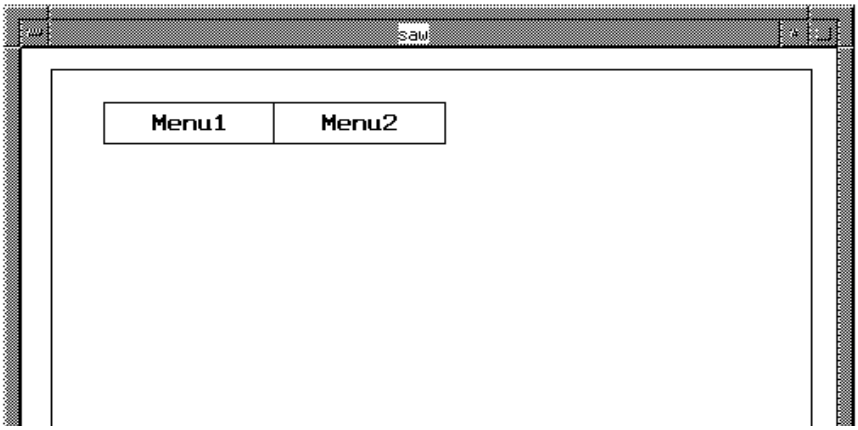

Figure 3-4 Menu1 and Menu2 Command Widgets

3.1.4 Duplicating the Menu2 Command Widget

Repeat the steps outlined in the previous section to make a duplicate of the **Menu2** widget. Position the new widget to the right of the **Menu2** widget, then change the label text to "Quit." Your workspace should now look like Figure 3-5.

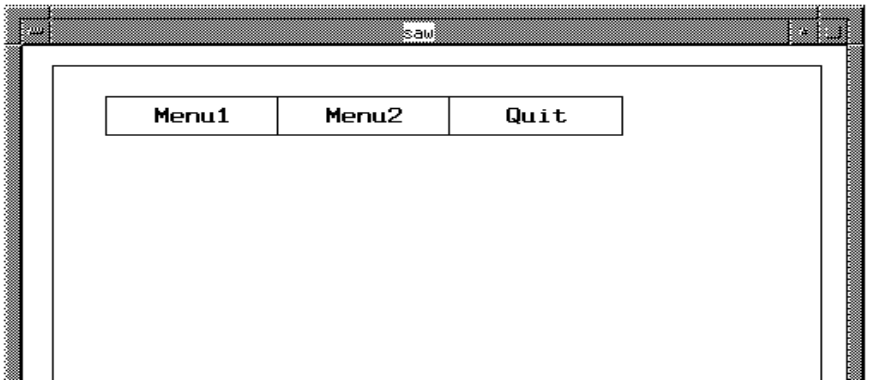

Figure 3-5 Top-Level Menubar

Note: At this point you should save your work. Refer to Section 2.3.1 for the use of the `Save as` menu entry from the `System` menu. You can use the file name `menubar`**.**

3.1.5 Adding a Form Widget

Select the `Form Widget` menu item from the **Draw** menu and place the form widget underneath the **Menu1** command widget. Resize it so it looks like Figure 3-6.

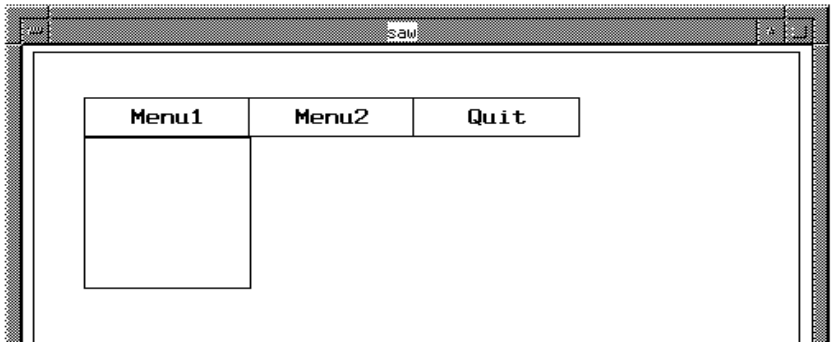

Figure 3-6 Form Widget Background for the Pull-Down Menu

The form widget will serve as the background for the pull-down menu and the additional command widgets.

3.1.6 Creating the Item_A Command Widget

In this section, you will create three more command widgets and place them on the new form widget. Select the **Menu1** widget and select the `Duplicate` menu item from the **Edit** menu. Move the new command widget so it fits inside the new form widget just underneath **Menu1**. Select the new widget and select the `Edit Attributes` menu item from the **Visualize** menu. Change the label text from “Menu1” to “Item_A.” The results are displayed in Figure 3-7.

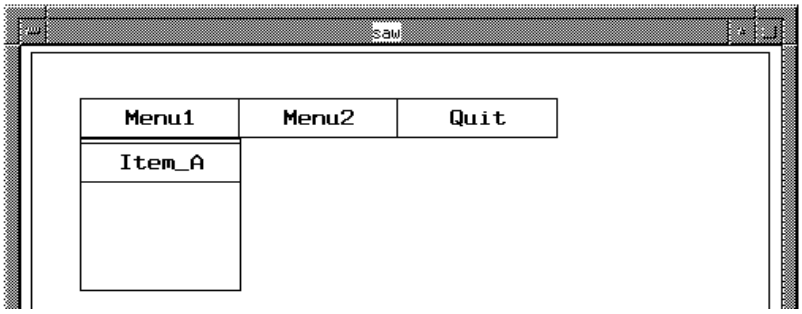

Figure 3-7 First Item for Pull-Down Menu1

3.1.7 Specifying the Method for the Item_A Widget

In this example, the same method is used for `Item_A` and `Item_B`. To save a step, specify the method for `Item_A` before duplication. Select `cw5` from the object list to display the **Attributes and Method**s menu. Select `method` to display the text box and type the following Slang code in the text box.

```
{ color_temp := background_color;
 background_color := foreground_color;
 foreground_color := color_temp; }
```

This method will reverse the default background and foreground colors when the widget is selected. When you are finished it should look like Figure 3-8. Once you have typed in the method, select `OK` in the text box to save the method.

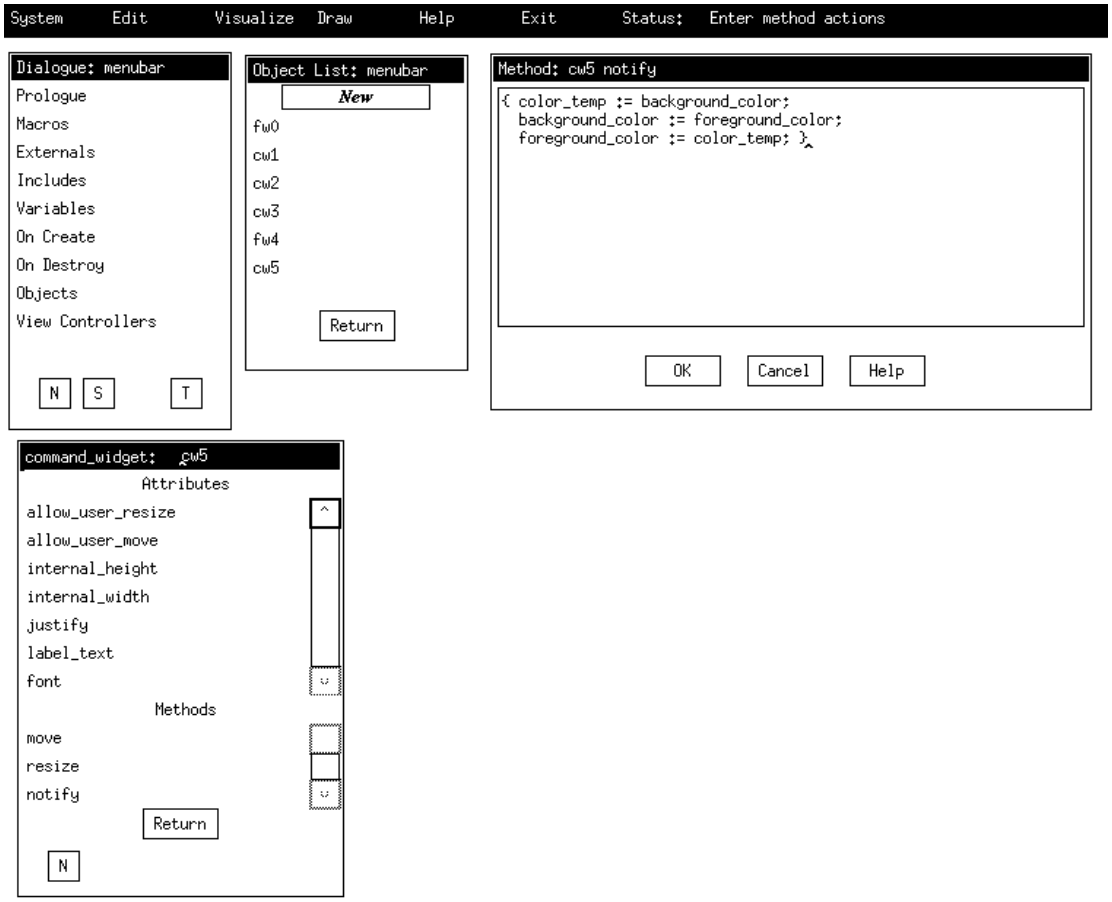

Figure 3-8 Method for Item_A Widget

3.1.8 Duplicate the Item_A Command Widget

Select the **Item_A** widget, then select `Duplicate` from the **Edit** menu. Place the new widget underneath the **Item_A** widget and change the label text from “Item_A” to “Item_B” via the **Edit Attributes** menu. To create another widget that will close the pull-down for **Menu1**, duplicate the **Item_A** widget again and place it below the **Item_B** widget. Change the label text to “Close Menu.” Since this widget has the same method as the **Item** widgets, you must change it. Select `cw7` (**Close Menu** widget) from the **Object** menu and select `method` from the **Attributes and Methods** menu to display the text box. Delete the Slang code left over from the previous widget and type the following Slang code:

```
{ display_menu1 := false; }co
```

This will cause the pull-down menu to close when the **Close Menu** widget is selected.

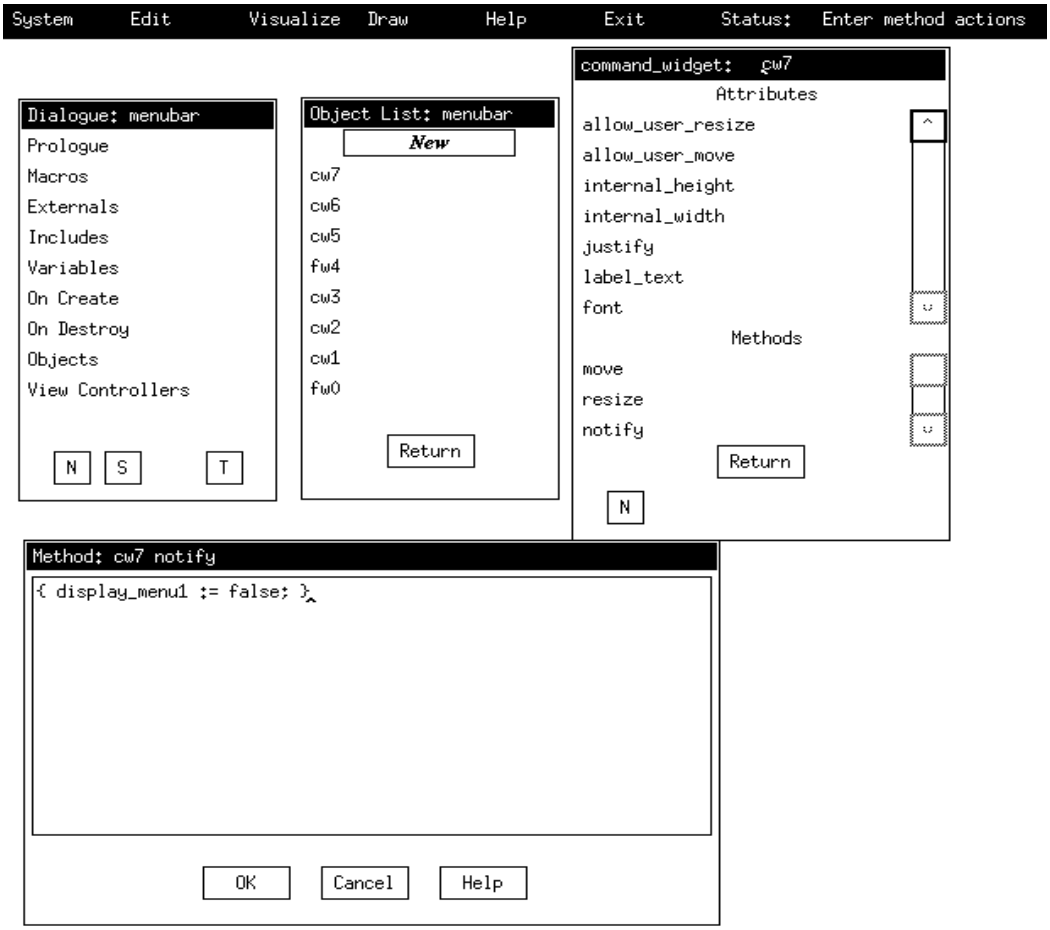

Figure 3-9 Method for Close Menu

Your workspace should now look like Figure 3-10.

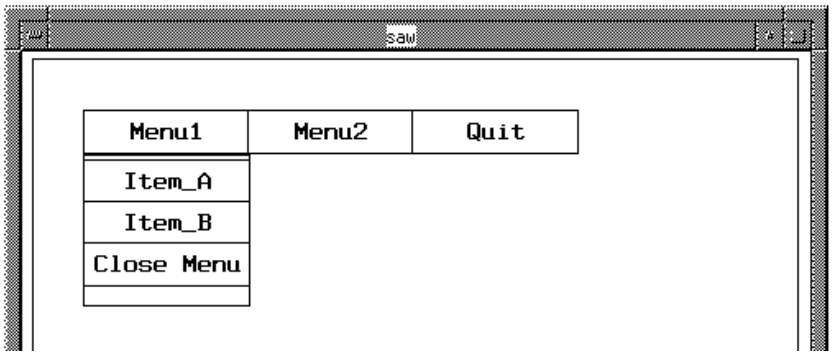

Figure 3-10 Items for Pull Down Menu1

Note: Use the `Save` menu item under the System menu to save your work again.

3.1.9 Creating Variables for the Menubar Example

Now that you have laid out a portion of the user interface for the menubar example, you must create several variables.

Note: Refer to Section 2.2.1 for more detailed information on creating variables.

Select the `Variables` entry on the **Dialogue** menu box shown in Figure 3-11. Select `New` to create the variable `Unnamed`. Then select the `Unnamed` entry in the **Variables** box to display the text entry box. Type in the variable name “display_menu” in the top of the text box as indicated by the arrow in the figure. In the text entry box type:

```
false;
```

Your workspace should look like Figure 3-11. Select `OK` to save the value for the `display_menu1` variable.

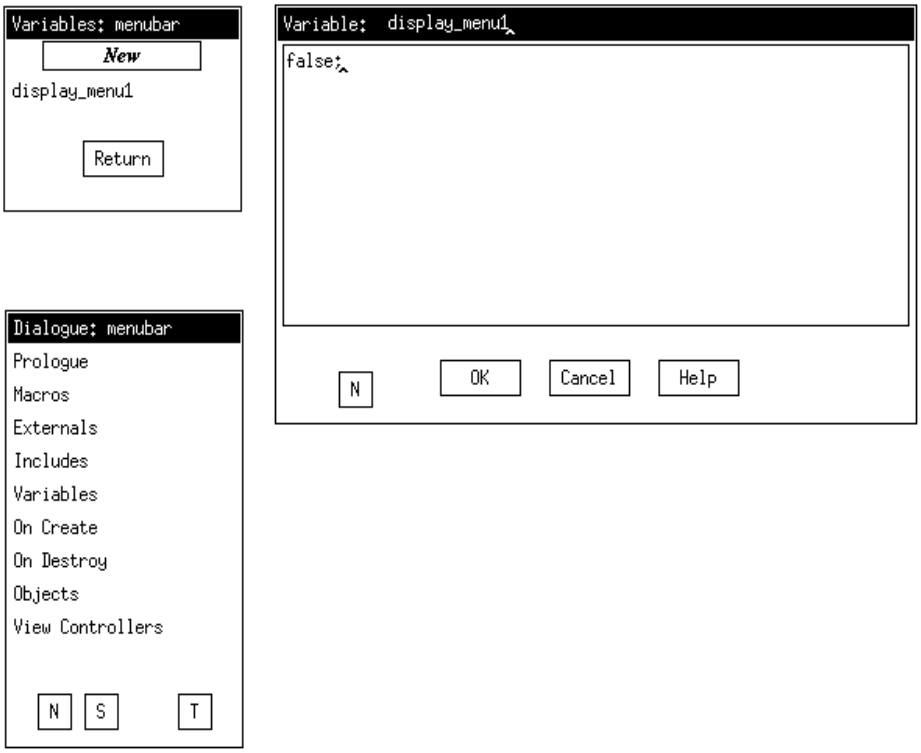

Figure 3-11 Variable Initialization for Display_menu1

Now select the `New` box to create another variable. Again, select `Unnamed` to bring up the text entry box. Type "`color_temp`" in the variable name slot and type the following in the text entry box to initialize the value to blank:

```
"";
```

After you have specified the initial value for `color_temp`, your screen should look like Figure 3-12. Remember to select `OK` to save the value for `color_temp`.

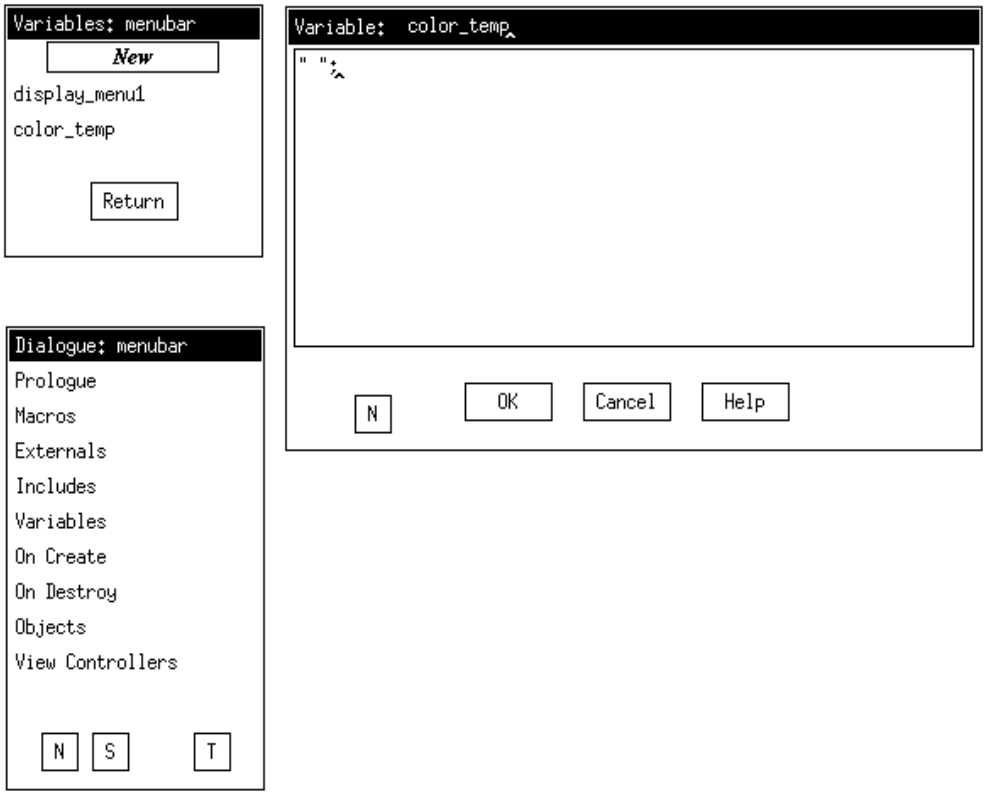

Figure 3-12 Variable Initialization for Color_temp

3.1.10 Specifying Methods for Widgets

To specify the method for the **Item_A** widget, you will use the steps outlined in Section 2.2.3. Select the `Objects` item in the **Dialogue** box as displayed in Figure 2-19.

3.1.11 Creating a View Controller

View controllers are created using the dialogue construct displayed in Figure 2-3. Once the view controller has been created, you can move the objects into it with the **Move** command.

4 Command Descriptions

This chapter presents descriptions of individual dialogue editor commands, ordered by where they appear in the menu hierarchy. These commands are used to perform editing, user interface layout, and dialogue specification tasks from within `ded` and can be accessed from the menubar. Commands may require or accept parameters and/or settings.

4.1 Parameters

Parameters carry information that must be provided before a command can be executed. It is often necessary to specify the appropriate parameters before commands can be selected. Components of the dialogue specification are commonly treated as parameters. Parameters can be selected in any order.

4.1.1 Sources and Targets

Source and target selection are examples of parameters that are commonly set by the user. The source parameter is the dialogue component on which the command will act.

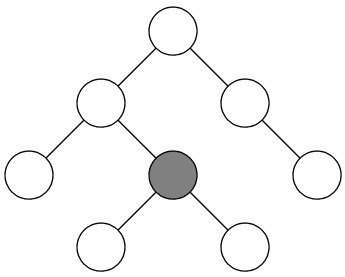

Figure 4-1 Dialogue Tree Node

Since the dialogue structure is represented as a tree, the source parameter may be either a single node, as in Figure 4-1, or an entire subtree, as in Figure 4-2.

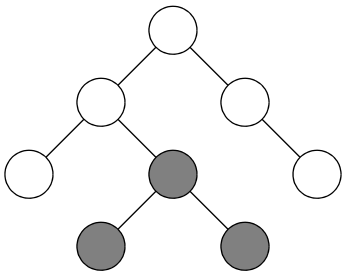

Figure 4-2 Dialogue Subtree

Figure 4-3 shows a view controller construct. The source parameter can be identified by selecting [N] on a specific dialogue construct. This identifies the construct as the source and sets the node/subtree parameter to “node.” As a result, only the construct is affected by the command. Selecting [S] causes the node/subtree parameter to be set to “subtree”; any ensuing command would then operate on the specified construct and all nested structures. Specific nodes can also be identified from a list by pressing the **Shift** key and selecting the object with the left mouse button.

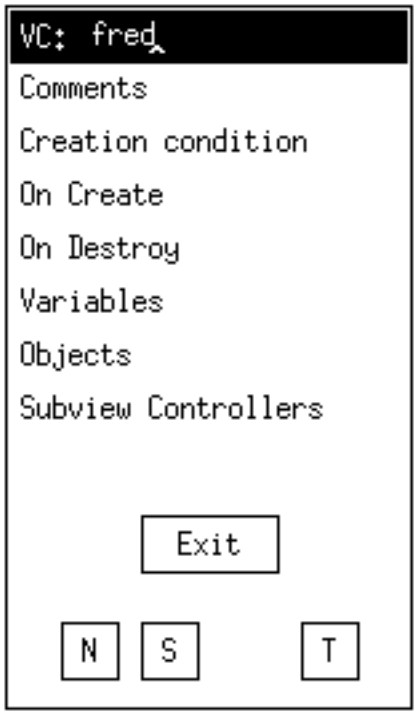

Figure 4-3 View Controller Construct

Targets are used to specify the location to which an item is copied or moved. The target parameter can be specified by selecting the [T] push button in the target construct.

Node, subtree, and target indicators are displayed in reverse video (on monochrome displays) when selected. They may be cleared by selecting a replacement source or target or by selecting the `Clear Operands` command from the **Edit** menu. Only one source or target may be identified at a time.

4.1.2 Object Types

Another common parameter is the object type. The object type must be identified to create either an object instance in the layout editor or an object template in the structure editor. Icons for toolkit objects appear in the toolkits "toolbox." The following icon is an example of a toolkit object:

The selected toolkit object is displayed in reverse video. Toolkit objects are selected with the left mouse button. Only one toolkit object may be selected at a time.

4.1.3 Settings

Settings are parameters prompted for by the command. Settings are entered through dialogue boxes. The following is an illustration of the dialogue box used by the `Save` command for requesting the name of the dialogue to be saved:

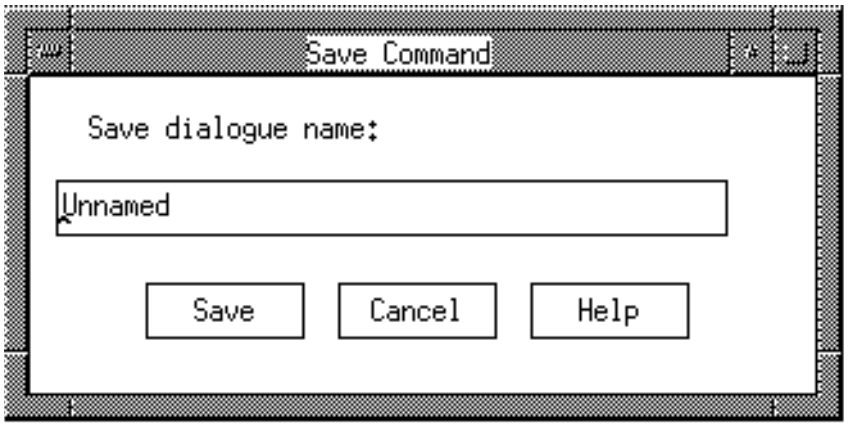

Figure 4-4 Save Dialogue Box

4.2 Dialogue Editor Commands

System commands are available from the **System** menu. They include commands that operate on the dialogue and typically involve transferring information between main memory and permanent storage. The command to quit from `ded` is also in the **System** menu.

New *[dialogue]*

Creates a new dialogue with the specified name. Only one dialogue can be loaded into the editor at a time.

Related Commands

Load, Save

Load *dialogue*

Loads an existing dialogue into the editor. Operates on the binary version of the dialogue, which is saved as an `.slb` file. Binary is the preferred format for saving and loading a dialogue since it preserves information that may be lost if the dialogue is saved in ASCII format.

Settings

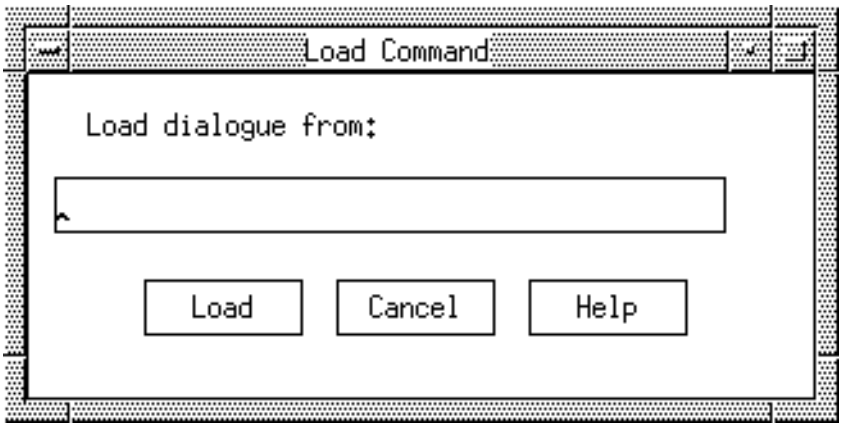

Figure 4-5 Load Command Dialogue Box

To load a dialogue, it is necessary to specify the name (but not the `.slb` extension) of the dialogue to be loaded.

Related Commands

Unload, New, Save

Save *[dialogue]*

Saves the current dialogue to disk. Save operates on the binary representation of the dialogue, saved as an .`slb` file. This is the preferred format for saving and loading a dialogue since it is the fastest format and it preserves information that may be lost if the dialogue is saved in ASCII format.

If the file is named, the Save command simply writes out the binary representation to a file of the same name as the dialogue. If the file is unnamed, you will be prompted to provide a dialogue name. The name of the current dialogue is also updated. See also the Save As... command for additional information.

Settings

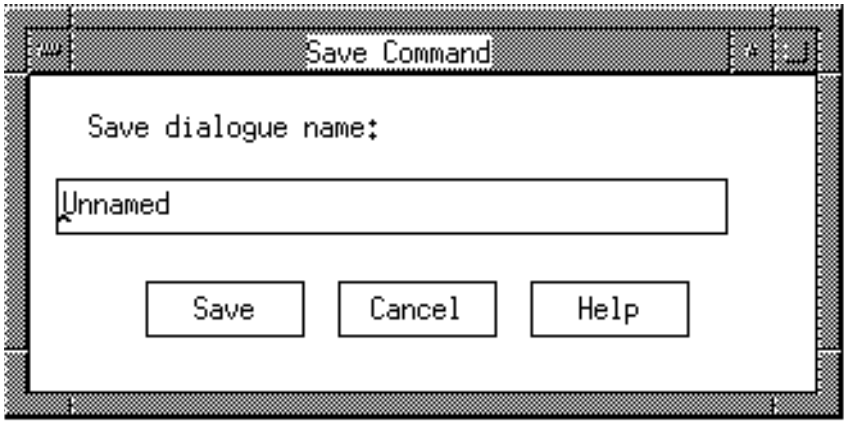

Figure 4-6 Save Command Dialogue Box

To save a dialogue it is necessary to specify a new dialogue name. When specifying the dialogue name, do not include the `.slb` extension since this is automatically added by the editor. The dialogue is saved in the working directory unless a path is specified.

Related Commands

Save As..., New, Load

Save As... *dialogue*

The Save As... command is the same as the Save command except that it always prompts for the dialogue name, even if the dialogue name is currently defined. The dialogue will be saved in binary format under the new dialogue name, followed by the `.slb` extension. The name of the current dialogue is also updated.

Import *dialogue*

The Import command can be used to import a Slang dialogue specification into the editor. Import operates on a Slang file saved as a `.sl` file. This command may be used to import existing Slang programs into the editor or to re-import exported files. We recommend the use of the Save and Load commands while maintaining existing dialogues within the editor. Using binary files is more efficient and guarantees that information is not lost.

Settings

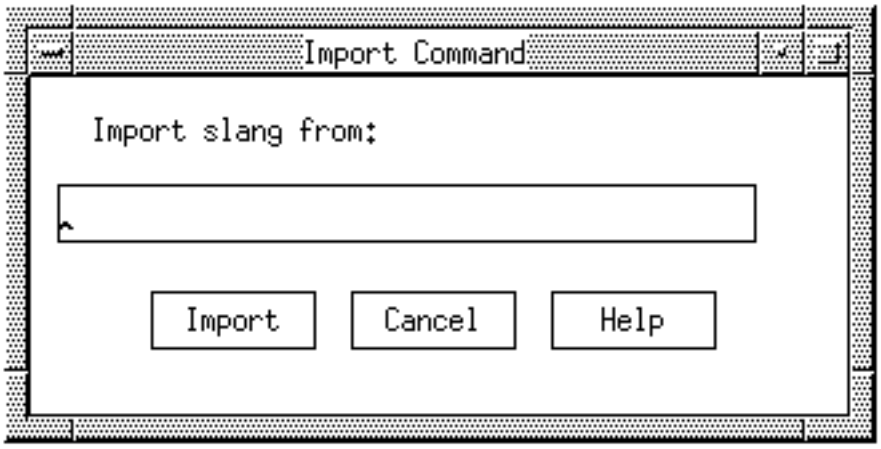

Figure 4-7 Import C Dialogue Box

To import a Slang dialogue specification into the editor, it is necessary to specify the name of the Slang file but not the `.sl` extension.

Related Commands

Export, Run

Export *dialogue*

The Export command can be used to generate a Slang specification of the current dialogue. This Slang specification can then be compiled and executed using the Run command or printed for hardcopy output. We recommend the use of the s**ave** and **load** commands while maintaining existing dialogues within the editor. Using binary files is more efficient and guarantees that information is not lost.

Settings

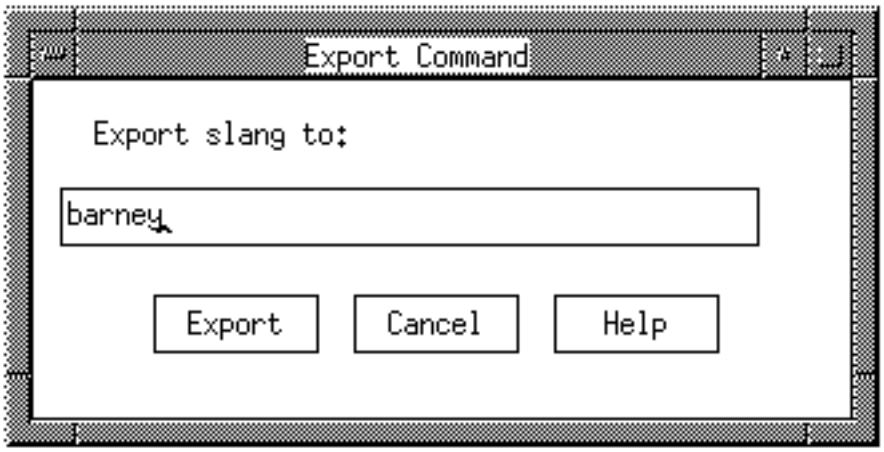

Figure 4-8 Export Command Dialogue Box

To export a dialogue, specify the name (but not the `.sl` extension) of the file in which the Slang specification is saved.

Related Commands

Import, Run

Run *dialogue*

The Run command can be used to execute a previously exported dialogue specification. The Run command invokes the Serpent `-clg` command for the specified dialogue. Output from the Run command will appear in the xterm window from which the editor was originally executed. If the dialogue compiles and executes successfully, the dialogue will appear in the window. This is actually a working copy of the dialogue, and behaves exactly as you have specified it. If you have not specified a manner in which to exit from your dialogue, you may need to invoke the Unix `kill -9` command to terminate the generated process.

Settings

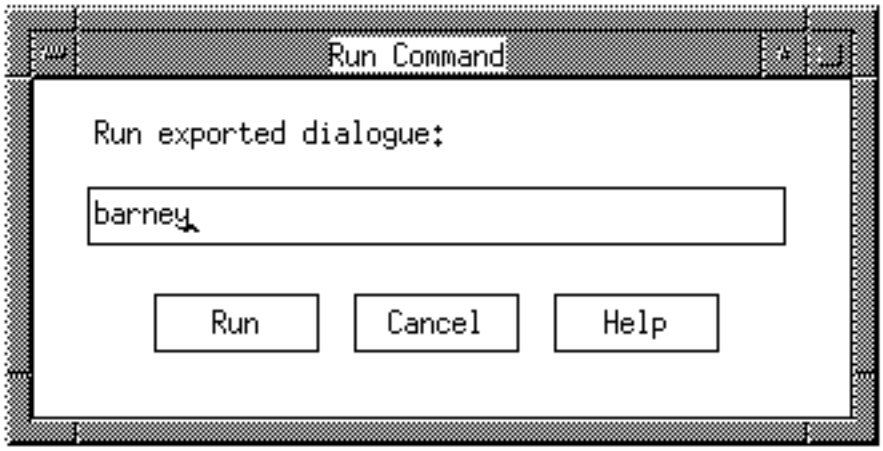

Figure 4-9 Run Command Dialogue Box

To run a Slang dialogue specification, specify the name (but not the `.sl` extension) of the Slang file.

Related Commands

Import, Export

Unload *dialogue*

Removes an existing dialogue from the editor without saving the changes. If the dialogue has been changed, you will be queried and warned that any existing changes will be lost. The editor remains active after removing the dialogue.

Related Commands

Load, New, Save

Serpent

The Serpent command displays information about Serpent development.

Quit

The Quit command allows the user to quit from the editor application. If the dialogue has been changed, you will be asked if you really want to quit without first saving these changes.

4.3 Edit Commands

Edit commands are used to modify existing dialogues or user interface prototypes and are available from the **Edit** menu or the **System** menu.

Edit Attributes of *object instance*

The Edit Attributes command is used to view and modify the attributes of an interaction object instance that has been created in the layout window. "Picking" an object and selecting the Edit Attributes command causes the edit attribute form (see Figure 4-10) for the object to be displayed. Newly created objects are automatically "picked" by the dialogue editor. Only one edit attribute form may be displayed at a given time.

Parameter

interaction object instance previously "picked"

Settings

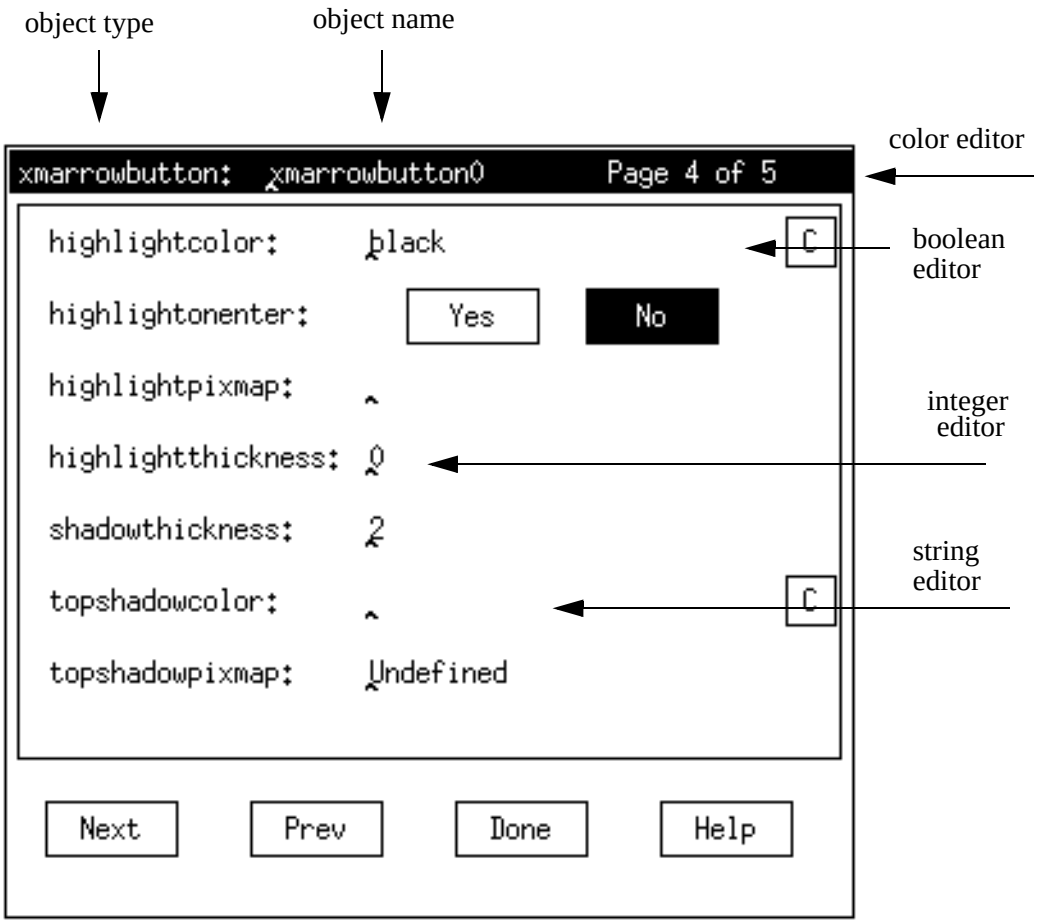

Figure 4-10 Edit Attributes Form

The edit attributes form has a number of components. Along the top bar of the form is status information reflecting the type and name of the object instance being edited.

The number of pages of an edit attributes form depends upon the number of attributes associated with a given object type. The **Next** and **Prev** buttons can be used to navigate through the various pages. The **Done** button exits from the form *without* saving any of the changes made in the form. The **Help** button provides information describing the object type.

Each line in the main edit area of the edit attributes form corresponds to a specific attribute associated with the object type. In Figure 4-10, for example, `shadow thickness` allows only valid integer values as input.

Related Commands

Import, Run

Copy *source* to *target*

Copies a view controller (with or without nested structures), object, or variable to a target location within the dialogue. This target location must be a view controller or the top-level dialogue structure. The Copy command is functionally equivalent to first duplicating the source construct(s) and then moving the duplicates; in either case the source is unmodified.

Parameters

- node or subtree to be copied
- target location

Related Commands

Duplicate, Delete, Move

Delete *source*

Deletes a view controller (with or without nested structures), object, or variable. When a node with nested structures is deleted, the nested structures can become “orphaned.” When this happens, the dialogue editor links the structures to the closest parent structure. This is illustrated in Figure 4-11.

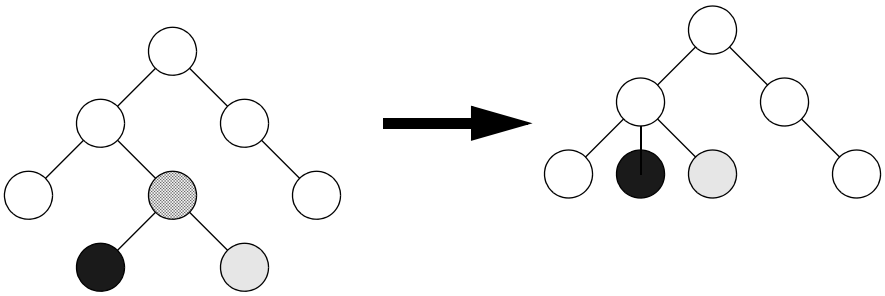

Figure 4-11 Deleting Nested Structures

When the cross-hatched node in the above diagram is deleted, the nested constructs are linked to the deleted node's parent.

Parameters

- node or subtree to be deleted

Related Commands

Copy, Duplicate, Move, Copy

Duplicate *source*

Duplicates a view controller (with or without nested structures), object template, object instance, or variable within the parent view controller or dialogue construct. The Duplicate command is useful for making multiple duplicates of a construct (such as an object) within the same context.

Parameters

- node or subtree to be copied
- target location

Related Commands

Duplicate, Delete, Move

Move *source* to *target*

Moves a view controller (with or without nested constructs), object, or variable to a target location within the dialogue. This target location must be a view controller or the top-level dialogue structure. The Move command is functionally equivalent to first copying the source and then deleting it. See the Delete command for more information on how the dialogue editor links potentially orphaned structures to the closest parent.

Parameters

- node or subtree to be copied
- target location

Related Commands

Duplicate, Move

Clear Operands *source, target, object type*

The Clear Operands command clears the source, target, and object type parameters, if set.

4.4 Layout Commands

Layout commands consist of three main types of commands. The first type are commands that bring up the layout area for different user interface toolkits that have been integrated into the system. Although Serpent allows multiple user interface toolkits to be used in a single dialogue, there is currently no way to connect interaction objects in the different toolkits directly. They must be connected indirectly through the dialogue specification.

The second collection of commands deals with the “visualization” of object templates in the dialogue structure. Constant values in the object template definition are used to create the corresponding object instance. Undefined or variable attributes are set to default values.

The third collection of commands are utility commands that can be used within the layout editor.

Motif Layout

The Motif Layout command brings up the Motif layout area and the Motif toolbox. The Motif layout area is used to create a prototype layout and the presentation of the different objects used in your user interface. The Motif toolbox contains icons representing the available Motif widgets.

Instantiate *view controller*

The Instantiate command is used to display a single instance of each object template defined within a view controller or view controller subtree, and to create them if they do not already exist.

Parameters

- view controller or subtree to be instantiated

Related Commands

Remove, Uninstantiate

Remove *view controller*

The Remove command affects those object instances that are associated with the object templates defined for a view controller or view controller subtree. It allows you to remove the object instances from the display without destroying them.

Parameters

- view controller or subtree from which the object instances should be removed

Related Commands

Instantiate, Uninstantiate

Uninstantiate *view controller*

The Uninstantiate command destroys the object instances associated with the object templates defined for a view controller or view controller subtree.

Parameters

- view controller or subtree to be uninstantiated

Related Commands

Remove, Instantiate

Grid Snap

The Grid Snap command allows the user to turn grid snap on or off within the layout editor. Grid Snap will align the position, width, and height of an object in a layout area when that object is created, moved, or resized so that the dimensions of the object are evenly divisible by the grid snap values. Grid Snap does not affect existing objects unless they are moved.

Grid Snap is useful in precisely aligning or sizing objects in the layout.

Related Commands

Grid Snap Values

Grid Snap Values

The Grid Snap Values command allows you to specify the grid snap values. These values are used to determine the grid intervals to which positional and size attributes should be snapped.

Grid snap calculations are based on the x, y, width, and height grid snap variables. These calculations are only made when Grid Snap is turned on.

Related Commands

Grid Snap

4.5 Toolkits

The **Toolkits** menu provides an alternate means of accessing the toolboxes for the different user interface toolkits that have been integrated into the system.

Athena Toolkit

The Athena Toolkit command brings up the Athena toolbox containing all the Athena interaction objects, or widget types.

The Athena “toolbox” window that contains the Athena toolkit can be moved, resized, or otherwise manipulated using window manager commands and capabilities. The area can also be scrolled to provide access to objects not currently displayed in the window.

An Athena interaction object type can be selected by clicking on the associated icon using the left mouse button and reset by re-selecting it or by executing the Clear Operands command from the **Edit** menu. The object type is used as a parameter both in the layout editor and in creating new object templates.

Related Commands

Motif Toolkit

Motif Toolkit

The Motif Toolkit command brings up the Motif toolbox containing all the Motif interaction objects, or widget types.

The Motif “toolbox” window that contains the Motif toolkit can be moved, resized, or otherwise manipulated using window manager commands and capabilities. The area can also be scrolled to provide access to objects not currently displayed in the window.

A Motif interaction object type can be selected by clicking on the associated icon using the left mouse button and reset by re-selecting it or by executing the Clear Operands command from the **Edit** menu. The object type is used as a parameter both in the layout editor and in creating new object templates.

Related Commands

Athena Toolkit

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release
Distribution Unlimited

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

4. PERFORMING ORGANIZATION REPORT NUMBER(S
CMU/SEI-91-UG-9

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Software Engineering Institute

6b. OFFICE SYMBOL (if applicable)
SEI

7a. NAME OF MONITORING ORGANIZATION
SEI Joint Program Office

6c. ADDRESS (City, State and ZIP Code)
Carnegie Mellon University
Pittsburgh PA 15213

7b. ADDRESS (City, State and ZIP Code)
ESD/AVS
Hanscom Air Force Base, MA 01731

8a. NAME OFFUNDING/SPONSORING ORGANIZATION
SEI Joint Program Office

8b. OFFICE SYMBOL (if applicable)
ESD/AVS

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F1962890C0003

8c. ADDRESS (City, State and ZIP Code)
Carnegie Mellon University
Pittsburgh PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Serpent: Dialogue Editor User's Guide

12. PERSONAL AUTHOR(S)
User Interface Project

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM TO	September 1991	74

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
Serpent, Serpent dialogue, UIMS, user interface management system, dialogue editor

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Serpent is a user interface management system (UIMS) that supports the development and implementation of user interfaces. This document provides guidelines for using the Serpent dialogue editor.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL
Charles J. Ryan, Major, USAF

22b. TELEPHONE NUMBER (Include Area Code)
(412) 268-7631

22c. OFFICE SYMBOL
ESD/AVS (SEI)

ABSTRACT —continued from page one, block 19

CMU/SEI-91-UG-1	Serpent Overview
CMU/SEI-91-UG-2	Serpent: System Guide
CMU/SEI-91-UG-3	Serpent: Saddle User’s Guide
CMU/SEI-91-UG-4	Serpent: Dialogue Editor User’s Guide
CMU/SEI-91-UG-5	Serpent: Slang Reference Manual
CMU/SEI-91-UG-6	Serpent: C Application Developer’s Guide
CMU/SEI-91-UG-7	Serpent: Ada Application Developer’s Guide
CMU/SEI-91-UG-8	Serpent: Guide to Adding Toolkits