Tutorial 3

This chapter demonstrates how to create a complete application with Devguide and Golit. In particular, it shows how to create projects and layered panes and how to use connections to handle events. It also shows how to modify generated callback functions, compile code, and run the completed application.

The sample application in this chapter consists of a text pane into which you can enter text and controls that set the appearance of the text. More specifically, it consists of a base window with:

- A text pane
- A control area with two buttons:
  - A menu button that switches between the two layered control areas
  - A button that displays a pop-up window
- Two layered control areas:
  - A control area with an exclusive setting that sets the font to Lucida or Rockwell
  - A control area with a checkbox that sets the font style to bold, italic, or both

The application provides a pop-up window, with:

- A control area with a slider that sets the text size.

The interface is shown in Figure 3-1 on page 22. Note that the control area that provides the font style checkbox is not visible.

3

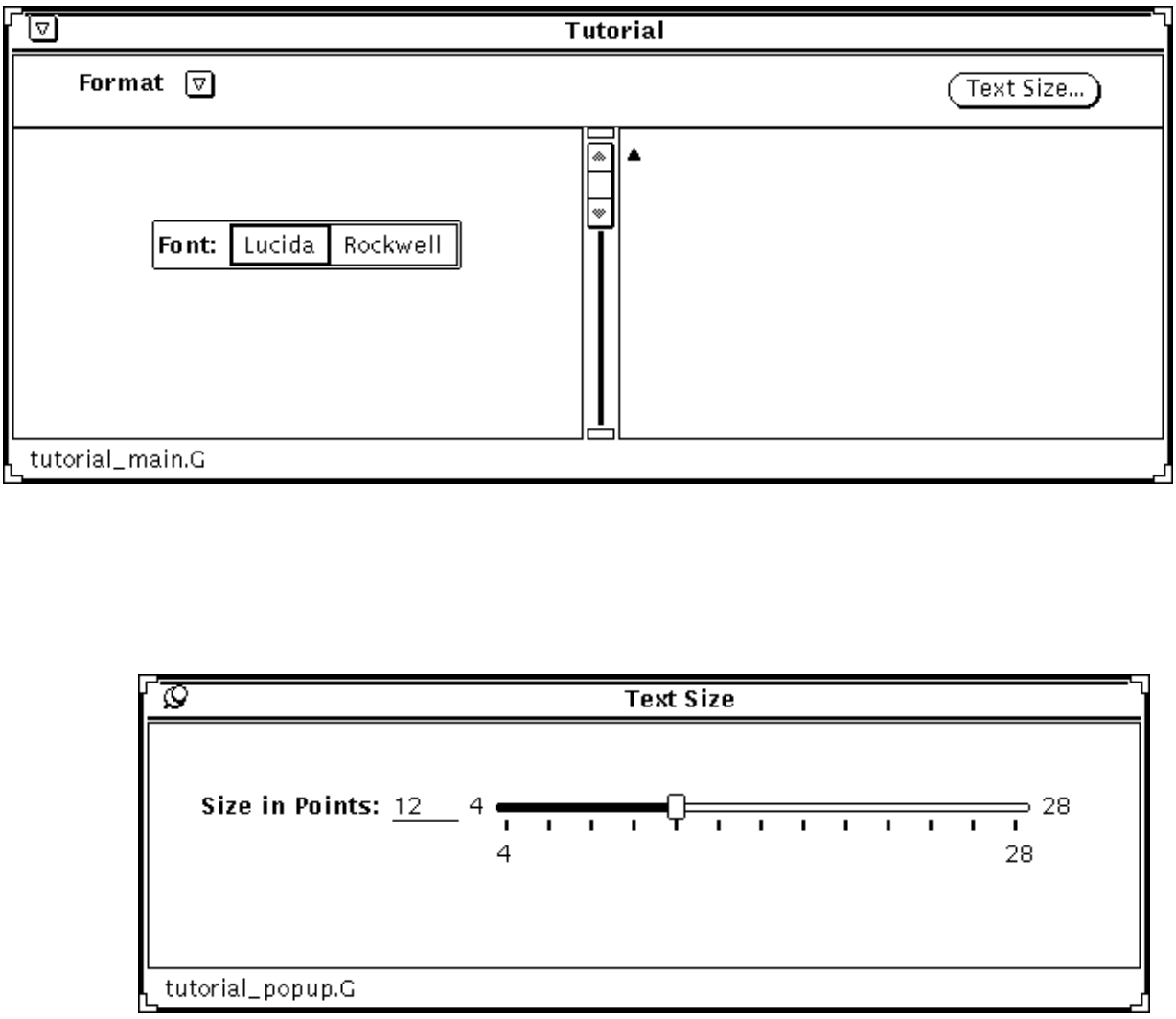

Figure 3-1 The Tutorial Application

Note – The example in this chapter occasionally diverges from the OPEN LOOK specification as it is outlined in the *OPEN LOOK Graphical UI Style Guidelines*. This is necessary to demonstrate a wide variety of Devguide features in a simple interface. When you develop your own applications, you should attempt to adhere to the specification.

Building an Interface

To create an interface with Devguide, you drag glyphs from Devguide's palette onto the workspace. Each glyph turns into an interface element. To customize an element's properties, open its property window, set property values, and click SELECT on the Apply button.

In this example, you will create a project with two interface (`.G`) files: one for the base window and one for the pop-up window. Note that the term, *interface*, refers to both the portion of your application stored in each `.G` file and to the entire application.

Creating a Directory for GIL and Project Files

When you create an application with Devguide, store all GIL (`.G`) and project (`.P`) files in one directory. Start Devguide and run Golit in that directory.

1. Create a new subdirectory in your home directory and name it `tutorial`.

```
% mkdir tutorial
```

2. Change directories from your home directory to `tutorial`.

```
% cd tutorial
```

3. Start Devguide.

```
% devguide &
```

3

Creating the Base Window Interface

All applications have one main base window which is visible (or iconic) when you first run an application. Figure 3-2 shows the completed base window interface for the sample application as it appears in the Browser.

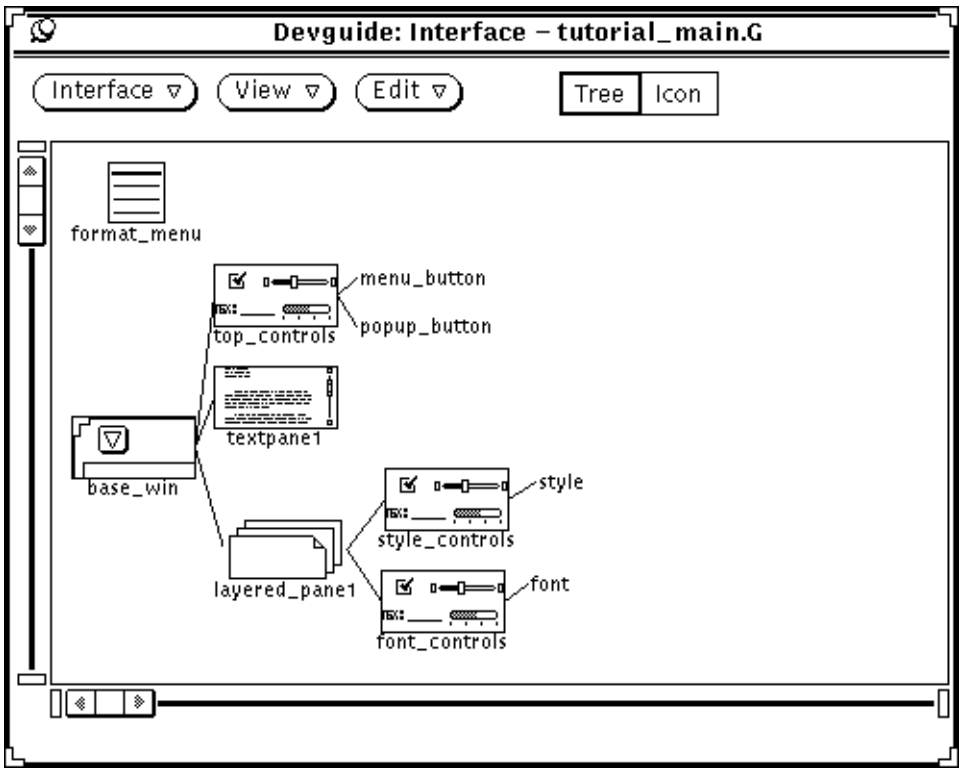

Figure 3-2 The Completed Base Window Interface Displayed in the Browser

Creating the Base Window

The first step in building an interface is to create a base window on the workspace. In this section you create a base window with layered control areas, a text pane, and various controls. You then save it in a GIL file.

1. **Drag the Base Window glyph from Devguide's palette onto the workspace and resize the base window.**
   Resize the base window so it is about three inches high and seven inches long.

2. **Customize the base window's properties.**
   Double-click SELECT on the base window's interior or footer to open the property window. Then, change the values in the fields:

Object	Property Field	Value
Base Window	Object Name	`base_win`
Base Window	Label	`Tutorial`

Note that the changes you make on the Properties window do not take effect until you click SELECT on the Apply button.

Adding a Control Area to the Base Window

You can place panes and control areas directly on a base or pop-up window. However, you *must* place control elements, such as buttons and sliders on a control area. A window can have more than one control area.

1. **Create a control area by dragging the Control Area glyph onto the top of the base window.**
   Place and resize the control area so it is about half an inch high and extends all the way across the top of the base window. This control area will contain the application's Format and Text Size buttons (see Figure 3-3).

2. **Customize the control area's properties as follows:**

Object	Property Field	Value
Control area	Object Name	`top_controls`

Adding Buttons to the top_controls Control Area

You create two different kinds of buttons for this application: an abbreviated menu button and a button to display a pop-up window.

1. **Place buttons in the top control area.**
   Drag the Button glyph and drop it at the left end of the control area. This button becomes the Format menu button. Later on, you will create a menu and attach it to this button.

   Drag the Button glyph again and drop it at the right end of the control area. This button becomes the Text Size... button, which displays the pop-up window.

2. **Customize the buttons' properties.**

Object	Property Field	Value
Left button	Object Name	`menu_button`
Left button	Label	`Format`
Left button	Type	Abbreviated Menu
Right button	Object Name	`popup_button`
Right button	Label	`Text Size...`

Figure 3-3 shows how the interface should appear when you have completed these steps.

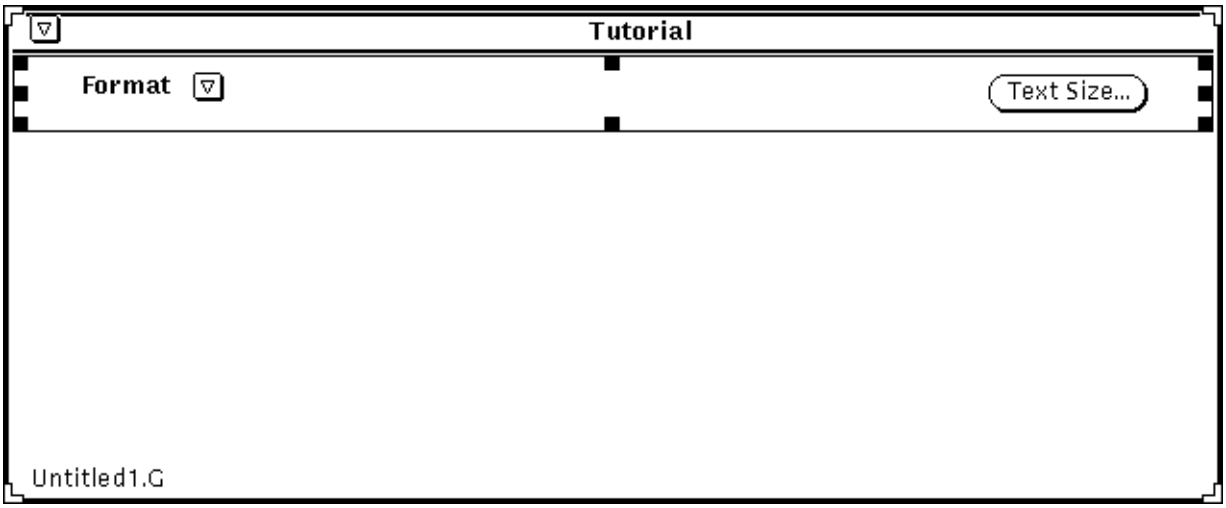

Figure 3-3 Example Interface with top_controls and Buttons

Adding a Text Pane to the Tutorial Base Window

To add a text pane to the base window, drag the Text Pane glyph onto the right side of the base window and resize it (see Figure 3-4). Use the default name, `textpane1`, for the text pane's Object Name.

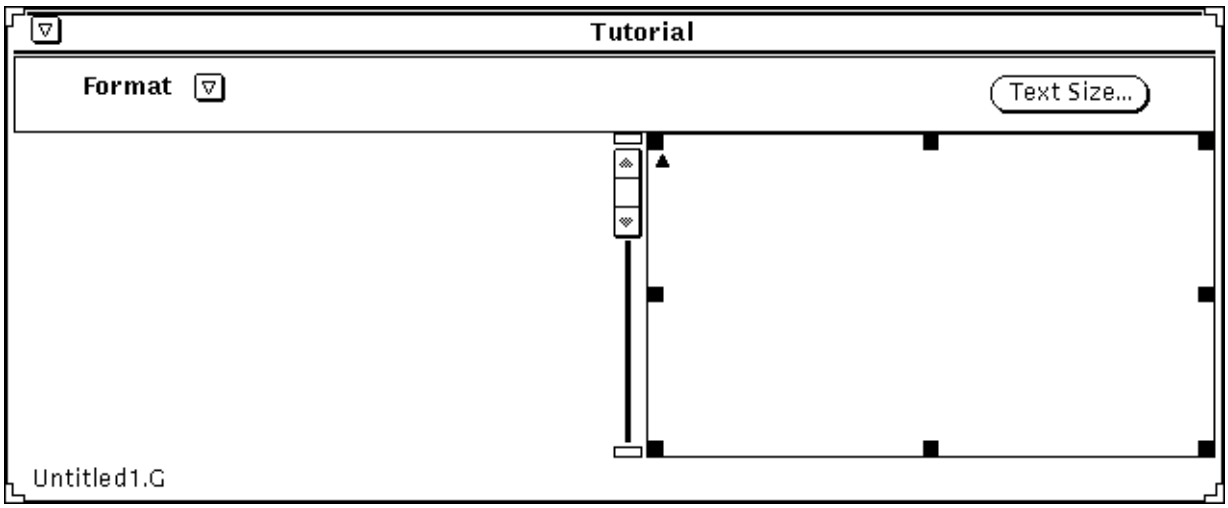

Figure 3-4 Example Interface with Text Pane

Creating Layered Control Areas

Devguide allows you to add layered panes to your application. You can layer control areas, text panes, or any other panes. In the completed application, only one pane is visible at a time. The application must include connections that make the desired pane visible and the other pane invisible. In this application, you will create connections to the menu, `format_menu`, to switch between the two control areas.

To create layered control areas:

1. **Drag the Control Area glyph onto the lower left control area of your base window.**
   Resize the control area to look like the one in Figure 3-5 on page 29.
2. **Drag the Control Area glyph onto the control area you just created**
   The new control area assumes the position and size of the original control area underneath it.

3. **Customize the control areas' properties:**

Object	Property Field	Value
controls1	Object Name	`style_controls`
controls1	Initial State	Invisible
controls2	Object Name	`font_controls`

4. The font_controls control area will be visible first when you run the completed application.

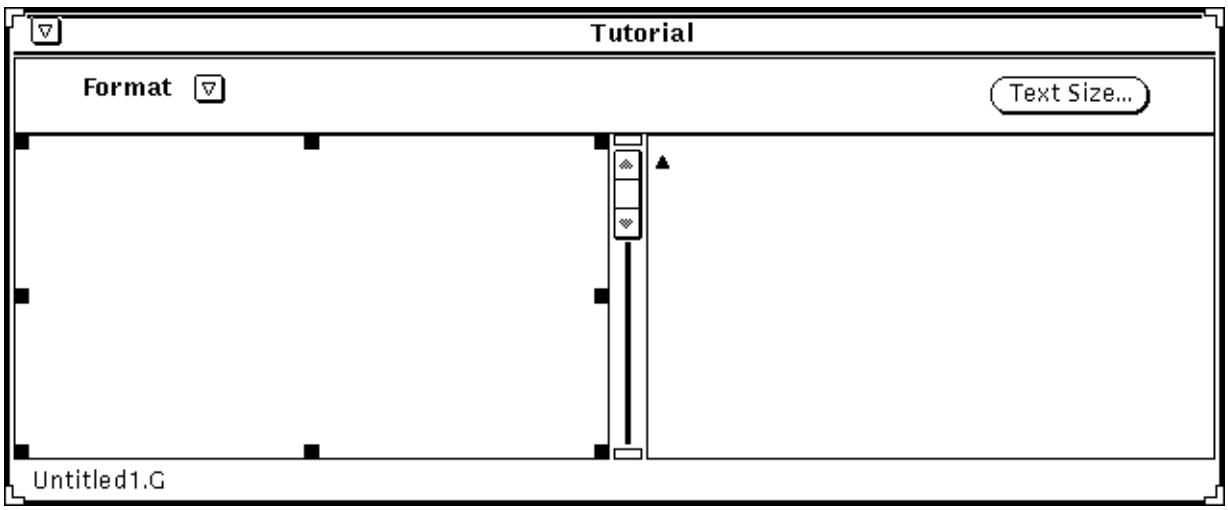

Figure 3-5 Example Interface with Layered Control Areas

Adding Controls to the Layered Control Areas

In this section, you will add controls that set the text font and style to the layered control areas. You will add an exclusive setting to font_controls and a checkbox setting to style_controls.

To add an exclusive setting to the font_controls control area:

1. **Drag the Exclusive Setting glyph onto the font_controls control area.**
   Settings can contain any number of choices; you will provide two, Rockwell and Lucida.

2. **Customize the setting's properties as follows:**

Object	Property Field	Value
Setting	Object Name	`font`
Setting	Label	`Font:`

3. **Customize the setting items' (Choices) properties.**
   The Choices scrolling list contains two choices. Set them as follows:

Object	Property Field	Value
Top Choice	Label	`Lucida`
Bottom Choice	Label	`Rockwell`

To add a checkbox setting to the style_controls area:

1. **Make the style_controls control area visible.**
   Select the font_controls control area. Choose Next Layer from the View menu. The style_controls control area will come to the front.

2. **Drag a Checkbox Setting glyph onto the control area.**
   Customize the checkbox setting's properties as follows:

Object	Property Field	Value
Checkbox Setting	Object Name	`style`
Checkbox Setting	Label	`Style:`

3. **Customize the individual checkboxes' (Choices) properties.**
   The Choices scrolling list contains two choices. Set them as follows:

Object	Property Field	Value
First Choice	Label	`Bold`
Second Choice	Label	`Italic`

Figure 3-6 shows the interface with style_controls in front; Figure 3-2 on page 24 shows it with font_controls in front.

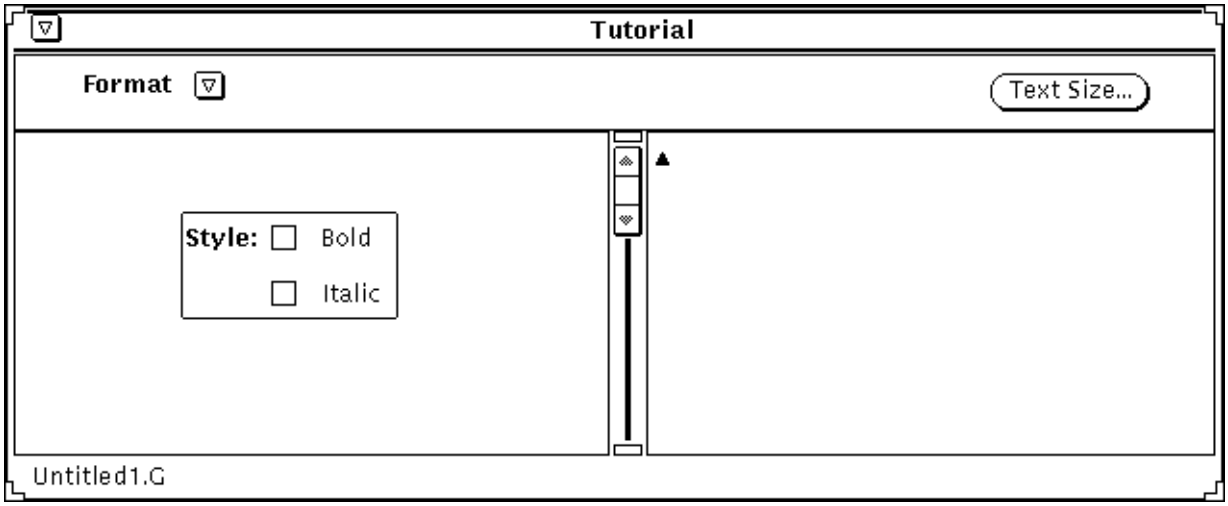

Figure 3-6 Example Interface with Style Controls Displayed

Saving the Tutorial Base Window Interface

To save the base window interface:

1. **Click MENU on Devguide's File button.**
2. **Choose Save As from the File menu to display the Save As file chooser.**
3. **Enter** `tutorial_main` **in the file chooser's Name field.**
4. **Click SELECT on the Save button**

Creating a Menu

You can attach menus to buttons, canvas panes, control areas, and scrolling lists. In this section, you create a menu and insert menu items using the Menu Editor window. In a later section, you will create a connection to make the menu appear when the Format menu button is pressed.

1. **Choose Menus... from Devguide's Properties menu.**
   This displays the Menu Editor window.

2. **Create a new menu.**
   Click SELECT on the Create button. This creates a new menu in the current interface, `tutorial_main.G`. Set the menu's properties as follows:

Object	Property Field	Value
Menu	Object Name	`format_menu`
Menu	Title	`Format`
Menu	Type	Exclusive

3. **Insert items into the exclusive menu.**
   Click SELECT on the Insert button to create a new menu item. Create two items with the following properties:

Object	Property Field	Value
First Menu Item	Label	`Font`
Second Menu Item	Label	`Style`

4. **Save the base window interface again.**
   Choose Save from Devguide's File menu. The menu you created is saved as part of `tutorial_main.G`.

Creating the Pop-up Window Interface

In this section, you create an additional interface and add a pop-up window, a control area, and a slider to it. Figure 3-7 shows the complete interface as it appears in the Browser.

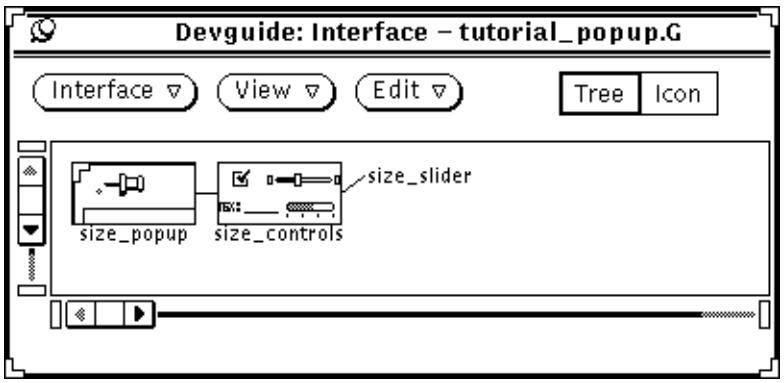

Figure 3-7 The Pop-up Window Interface Displayed in the Browser

Creating the Pop-up Window

1. **Choose New Interface... from Devguide's File menu.**
   The Interface Browser is displayed on the workspace. Note that anything you drag onto the workspace is part of the new interface until you create another interface or modify a previously made interface. Icons for new interface elements appear in the Browser's window.

2. **Drag the Pop-up Window glyph onto the workspace and resize the pop-up window.**
   Use the resize corners to stretch the window so that it is about two inches high and six inches long.

   Customize the pop-up window's properties as follows:

Object	Property Field	Value
Pop-up window	Object Name	`size_popup`
Pop-up window	Label	`Text Size`
Pop-up window	Window Parent	`base_win`

Adding a Control Area to the Pop-up Window

1. **Drag the Control Area glyph onto the pop-up window and resize the control area.**
   Resize the control area to fill the pop-up window.
2. **Customize the control area's properties as follows:**

Object	Property Field	Value
control area	Object Name	`size_controls`

Adding a Slider to the Control Area

To enable the user to set the text font size, add a slider to the control area. Modify the slider's properties so that the user can specify any font size between 4 and 28.

1. **Drag the Slider glyph onto the pop-up window's control area.**

2. **Customize the slider's properties as follows:**

Object	Property Field	Value
slider	Object Name	`size_slider`
slider	Label	`Size in Points:`
slider	Width	`240`
slider	Range Min	`4`
slider	Range Max	`28`
slider	Tick String Min	`4`
slider	Tick String Max	`28`
slider	Ticks	`13`
slider	Initial Value	`14`

Saving the Pop-up Window Interface

To save the pop-up window interface:

1. **Click MENU on Devguide's File button.**
2. **Choose Save As from the File menu to display the Save As file chooser.**
3. **Enter** `tutorial_popup` **in the file chooser's Name field.**
4. **Click SELECT on Save.**

Creating and Saving a Project

If you have more than one interface file in your application, as in this tutorial, create a project. To create a project consisting of the base and pop-up window interfaces:

1. **Choose Project... from Devguide's File menu to display the Project Organizer window.**
   The window displays a file icon for each GIL file currently loaded in Devguide. Icons should appear for `tutorial_main.G` and `tutorial_popup.G`.
2. **Choose Save As from the Project menu.**
3. **Enter** `tutorial` **in the Save As file chooser's Name field.**
4. **Click SELECT on Save.**
   This produces a project file named `tutorial.P`.

Setting up Connections

Devguide provides two kinds of connections that handle events in your application:

- Predefined connections, which generate complete callback code
- CallFunction connections, which generate callbacks that you insert custom code into

In this section you establish predefined connections to:

- Display the Format menu when the Format menu button is pressed

- Switch between the layered control areas when an item is chosen from the Format menu
- Display the Text Size pop-up window when popup_button button is pressed

You also establish CallFunction connections to:

- Change the text font size whenever the value of size_slider is changed
- Set the text font type and style whenever the controls in font_controls or style controls are changed

You must write the code to execute the CallFunction connections.

To set up connections, use the Connections Manager window. Choose the appropriate Source and Target menu items and select the Source and Target you want from the scrolling lists. Choose the When and Action menu items for the connection. In some cases, you must enter an argument for the Action. Click SELECT on the Connect button to establish the connection.

You can open the Connections Manager window from Devguide's Properties menu, from an object's property window (using the Connections... button), or by using drag-and-link.

Note – You can resize the Connections Manager window to display the full item names in the scrolling lists.

Creating a Connection to Display the Format Menu

To create the connection that makes the Format Menu appear when menu_button is pressed:

1. **Open the Menu Editor window.**
   Choose Menus... from the Properties Menu
2. **Click SELECT on the Connections... button in the top section of the Property sheet.**

3. **Choose Buttons from the Connections Manager Source menu and Menus from the Target menu.**
   Set up the following connection:

Source	Target	When	Action
menu_button	format_menu	Menu	Show

Be sure to click SELECT on the Connect button to create the connection.

Creating Connections to Switch between Layered Control Areas

To switch between the layered control areas, you need to create four separate connections to do the following:

- Show style_controls when Style is chosen
- Hide font_controls when Style is chosen
- Show font_controls when Font is chosen
- Hide style_controls when Font is chosen

To create the connections (in the Connection Manager), choose Menu items from the Source menu and Control Areas from the Target menu. The item names that appear in the scrolling lists are prefixed with the menu name (for example, format_menu.Font). Note that you may have to resize the Connections Manager Window so that you can see the entire names.

Create the following connections:

Source	Target	When	Action
Font	font_controls	Notify	Show
Font	style_controls	Notify	Hide
Style	style_controls	Notify	Show
Style	font_controls	Notify	Hide

Creating a Connection to Display the Text Size Pop-up Window

You can use drag and link to create a connection that displays the pop-up window when you click SELECT on the Text Size menu button:

1. **Click SELECT on the Text Size button.**

2. **Press the Meta key and drag the pointer from the button to the pop-up window's header.**
   The pointer turns into a cord with a plug at the end.

3. **Release the SELECT button.**
   The Connection window will appear if it is not already on the screen. Drag and link sets the Text Size button and pop-up as the default Source and Target objects. Check that these are set correctly.

Source	Target	When	Action
popup_button	size_popup	Notify	Show

4. **Click SELECT on the Connect button to create the connection.**

Creating a Connection to Set the Text Size with the Slider

To create a connection that enables the slider to set the text size, choose Sliders from the Source and Target menus and specify the following:

Source	Target	When	Action	Arg
size_slider	size_slider	Notify	CallFunction	`set_textsize`

Creating Connections to Set the Font and Style

These connections will all be CallFunction Connections. Note that you must provide two connections for each checkbox: one for when it is selected (to set the font to bold or italic) and one for when it is deselected (to set the font back to normal). To set up the connections, choose Setting Items from the Source and Target menus (in the Connections Manager). The item (checkbox) names displayed in the scrolling lists are prefixed with the setting name (for example, font.Lucida). Specify the connections as follows:

Source	Target	When	Action	Arg
Lucida	Lucida	Notify	CallFunction	`set_lucida`
Rockwell	Rockwell	Notify	CallFunction	`set_rockwell`
Bold	Bold	Notify	CallFunction	`set_bold`
Italic	Italic	Notify	CallFunction	`set_italic`
Bold	Bold	Unselect	CallFunction	`set_unbold`
Italic	Italic	Unselect	CallFunction	`set_unitalic`

Experimenting with Test Mode

You have created several predefined connections that work in Test mode. Try them out to see how parts of the tutorial user interface work. First enter Test mode. Then display the Format menu, scroll through the layered panes, and display the Text Size pop-up window.

1. **Enter Test mode by clicking SELECT on Test.**
   The Test choice is highlighted. All the predefined connections in your application will work. The only connections that will not work are CallFunction connections.

2. **Put the pointer over the text pane you created and enter some sample text.**

3. **Press MENU on the Format menu button.**
   This displays the Format menu.

4. **Choose Style from the Format menu.**
   You now see the checkbox setting that allows you to choose Bold, Italic, or both. When you click SELECT on a choice, that choice highlights. You cannot change the text style in Test mode, because the underlying connections are CallFunction connections. See "Setting up Connections" on page 35 for more information about connections.
5. **Choose Font from the Format menu.**
   You now see the Font exclusive setting. You cannot change the text font in Test mode.
6. **Click SELECT on the Text Size button to display the Text Size pop-up window.**
   This displays the slider you created. You cannot change the font size in Test mode.
7. **Click SELECT on Build to reenter Build mode.**
8. **Unload the previously-saved base and pop-up window interface files.**

Generating Interface Source Code

To generate the interface source code, first make sure you have saved all your changes. Then run the `golit` command on your project file. Golit lists the files as it reads and writes them:

```
% golit -p tutorial
golit: reading tutorial.P
golit: reading tutorial_main.G
golit: writing tutorial_main_ui.h
golit: writing tutorial_main_ui.c
golit: writing tutorial_main_stubs.c
golit: reading tutorial_popup.G
golit: writing tutorial_popup_ui.h
golit: writing tutorial_popup_ui.c
golit: writing tutorial_popup_stubs.c
golit: writing tutorial.c
golit: writing tutorial.h
golit: writing Makefile
```

Modifying the Golit-Generated Code

In this section you will alter the Golit-generated files to make the CallFunction connections work. The callback functions Golit generates for these connections contain print statements that you will replace with custom code. You will also create a function called `set_font()` which the `set_rockwell()`, `set_lucida()`, `set_italic()`, `set_bold()`, `set_unitalic()`, `set_unbold()`, and `set_textsize()` callbacks will use to change the text pane's font, style, and size.

Creating a Header File

To register the user's choice of Bold and/or Italic, and Rockwell or Lucida and font size, you must create a global structure. To do this, create a separate header file, `myheader.h`, that contains the following declaration:

```
struct characterformat {
          String font;
          Boolean bold;
          Boolean italic;
          String size;
} characterformat;
```

Add the following line below the other `#include` lines in the `tutorial.c`, `tutorial_main_stubs.c`, and `tutorial_popup_stubs.c` files:

```
#include "myheader.h"
```

Customizing the tutorial.c File

To initialize the `characterformat` structure, add the following code after the beginning of the `main()` program in `tutorial.c`:

```
characterformat.font = "lucidasans";
characterformat.bold = FALSE;
characterformat.italic = FALSE;
characterformat.size = "14";
```

This sets the `characterformat` structure to the normal default font settings.

The CallFunction callbacks that you set up in Devguide must reset the textpane font to reflect the user's choice of font style and size. It is easiest to do this by calling a common function that sets the font to the specifications in the `characterformat` structure. Define this function, `setfont()`, at the end of the `tutorial.c` file as follows:

```
/* Sets font to that specified by characterformat struct */
void
setfont(widget)
Widget widget;
{
    Arg arg;
    char fontname[64]; /* array to store font name */
    struct characterformat *cf = &characterformat;

    /* Get Widget id for the text pane */
    Widget target = GolitNameToWidget(widget,
        "Tutorial_main_textpane1");

    /* Concatenate font descriptions into font name */
    sprintf (fontname, "%s%s%s%s-%s",
        cf->font,
        ((cf->bold || cf->italic) ? "-" : ""),
        (cf->bold ? "bold" : ""),
        (cf->italic ? "italic" : ""),
        cf->size);

    /* Set text pane font resource to font name */
    XtVaSetValues(target, XtVaTypedArg, XtNfont, XtRString,
         fontname,strlen(fontname) + 1, NULL);
}
```

The `setfont()` function uses `GolitNameToWidget()` to retrieve the widget id of the textpane. The widget name (`Tutorial_main_textpane1`) that you pass to `GolitNameToWidget()` is constructed by joining the textpane object's name to the interface name.

`setfont()` creates a fontname by compiling the description in the `characterformat` structure. It starts with the selected font. If the `bold` or `italic` variables are TRUE, it adds `bold` or `italic` to the name. Finally, it

adds the font size. For example, if the user has selected Rockwell, checked the bold checkbox, and specified a point size of 20 with the slider, `setfont()` creates the font name, `rockwell-bold-20`. `setfont()` uses `XtVaSetValues()` to set the textpane font to this font name.

Customizing the tutorial_main_stubs.c File

You created CallFunction connections for both checkbox and exclusive settings earlier in this tutorial. Since these connections were only for the base window, Golit generates the callbacks for them in the `tutorial_main_stubs.c` file. These callbacks have the names that you specified in Devguide and appear along with callbacks for the predefined connections you specified in Devguide.

Each of the CallFunction callbacks must set a field of the `characterformat` structure to reflect user input. They must then invoke `setfont()` to set the textpane font to that specified by the changed `characterformat` structure.

Modify the checkbox callbacks to set the font characteristics as follows:

```
void
set_bold(widget, clientData, callData)
         Widget widget;
         XtPointer clientData, callData;
{
         characterformat.bold = TRUE;
         setfont(widget);
}

void
set_unbold(widget, clientData, callData)
         Widget widget;
         XtPointer clientData, callData;
{
         characterformat.bold = FALSE;
         setfont(widget);
}

void
set_italic(widget, clientData, callData)
         Widget widget;
         XtPointer clientData, callData;
{
         characterformat.italic = TRUE;
         setfont(widget);
}

void
set_unitalic(widget, clientData, callData)
         Widget widget;
         XtPointer clientData, callData;
{
         characterformat.italic = FALSE;
         setfont(widget);
}
```

Modify the callbacks for the exclusive settings to set the font to `rockwell` or `lucida` as follows:

```
void
set_rockwell(widget, clientData, callData)
         Widget widget;
         XtPointer clientData, callData;
{
         characterformat.font = "rockwell";
         setfont(widget);
}

void
set_lucida(widget, clientData, callData)
         Widget widget;
         XtPointer clientData, callData;
{
         characterformat.font = "lucidasans";
         setfont(widget);
}
```

Customizing the tutorial_popup_stubs.c File

In Devguide you set up a connection that is activated by a change in the slider position. Golit generates the callback, `set_textsize()`, for this connection in `tutorial_popup_stubs.c`. Modify this callback so that it retrieves the new slider value from the `callData` variable passed to the function. Add code to convert that value to a string, and set the `size` variable of `characterformat` structure to it. Add a call to the `setfont()` function to apply the new font size:

```
void
set_textsize(widget, clientData, callData)
         Widget widget;
         XtPointer clientData, callData;
{
 /* Get slider value from callData  */
 int size = *(int * )callData;

 /* Convert the slider value to string  */
 sprintf(characterformat.size, "%1d", size);

 /* Apply characterformat to the textpane  */
 setfont(widget);
}
```

Compiling and Running the Tutorial Application

To compile code for your sample application, you enter `make` in the shell tool. `make` uses the Golit-generated `Makefile` to compile code.

1. **Run `make` to compile and link the application.**

```
% make
```

   An executable file named `tutorial` is produced, assuming you have not made any errors.

2. **Run `tutorial` and test it.**

```
% tutorial
```