GXV Tutorial 3

This chapter demonstrates how to create a complete application with Devguide and GXV. 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 consists of a text pane which you can enter text into and controls that modify the appearance of the text. More specifically, it consists of a base window with:

- A text pane which you can enter text into.
- Two layered control areas:
  - A control area with an exclusive setting that sets the font to Roman or Lucida.
  - A control area with a checkbox setting that sets the font style to bold, italic, or both.
- A third control area with two buttons.
  - A menu button that switches between the two layered control areas.
  - A button that displays a popup window.

The application provides a popup window, with:

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

The application is shown in Figure 3-1 on page 24. Note that the control area that provides the font style checkbox is not visible. Figure 3-6 on page 33 shows the base window interface with the style checkbox setting in front.

3

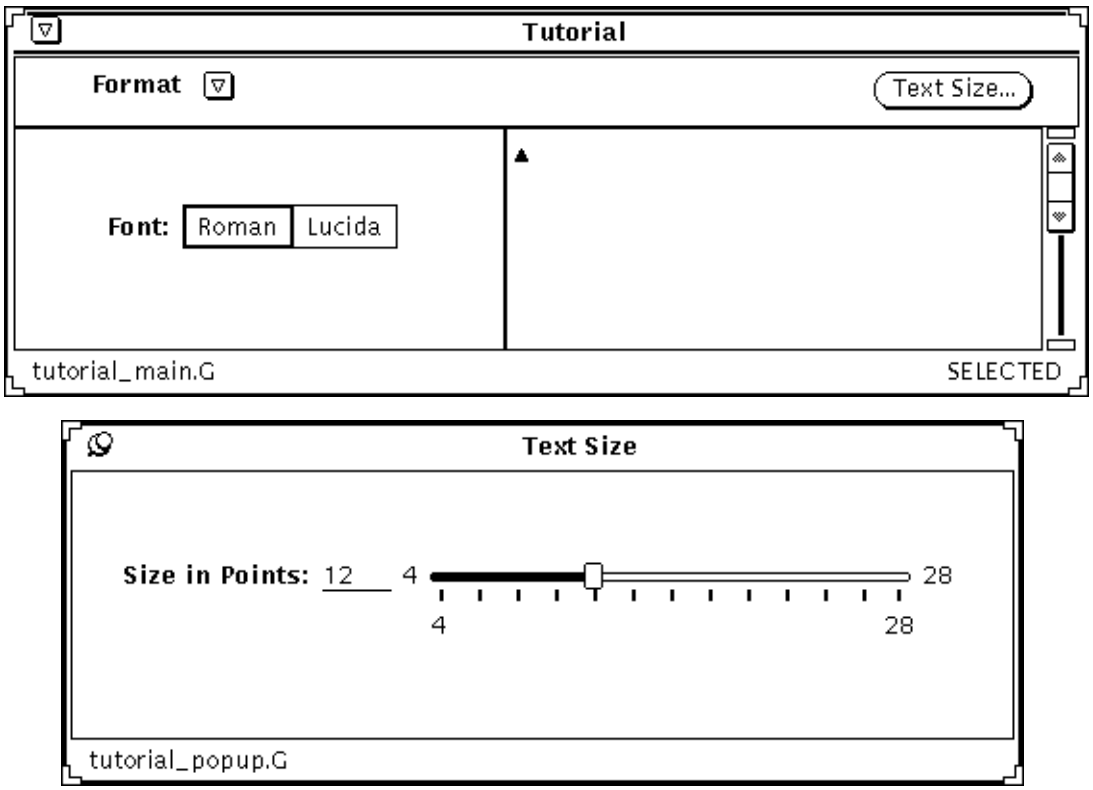


Figure 3-1 Tutorial Application

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 the Interface

To create an interface with Devguide, you drag glyphs from Devguide's palette onto the workspace. Each glyph turns into an interface element. You can lay elements down one above another or one next to another to assemble the

interface you want. You can also layer panes in your interface. To customize an element's properties, open its property window, set property values, and click SELECT on the Apply button.

You can store your application in one or more GIL files. Create a project file if you use more than one window; store each window in a separate GIL file. Note that the term *interface* refers to both the portion of your application stored in each GIL file, as well as to the entire application itself.

In this example, you will create a project with two interface (GIL) files: one for the base window and one for the pop-up window. You create the first interface by simply dragging glyphs onto the workspace. To create an additional interface file, choose New Interface... from Devguide's File menu; then assemble the interface.

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 gxv in that directory.

1. **Create a new subdirectory of your home directory and name it** `Tutorial`**.**

```
% mkdir Tutorial
```

   You can delete this subdirectory when you are done with the tutorial. `Tutorial` should be empty when you start.

2. **Change directories from your home directory into** `Tutorial`**.**

```
% cd Tutorial
```

3. **Start Devguide.**

```
% devguide &
```

3

Creating the Base Window Interface

All applications have one main base window that is visible (or iconic) when you first run an application.

Figure 3-2 shows the completed base window interface as it appears in the Interface Browser. (To view this browser after you have saved the interface, choose View from Devguide's File menu, then choose the interface file name from the Interfaces submenu.)

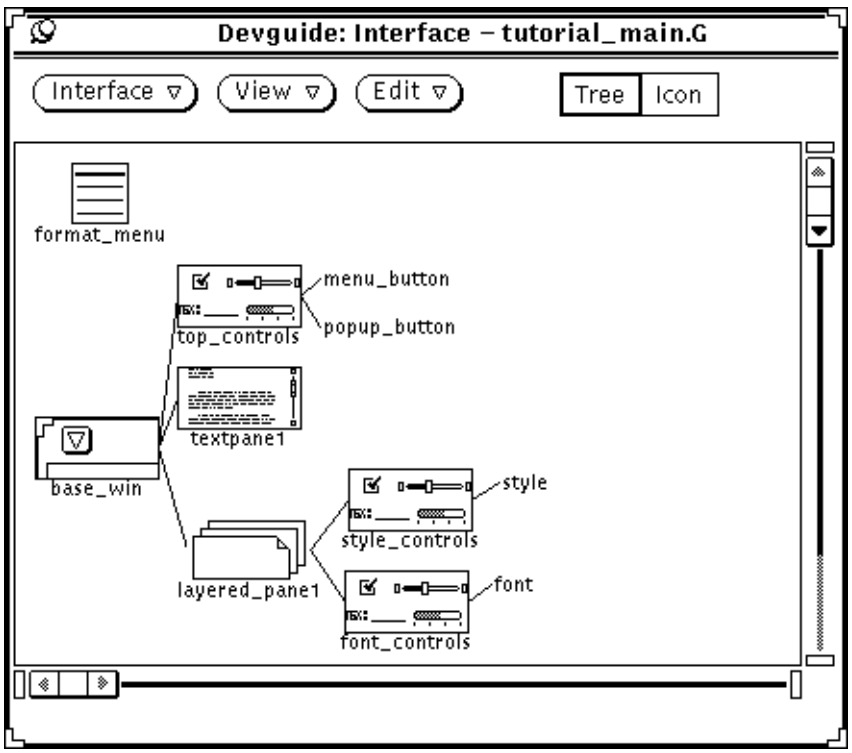


Figure 3-2 Base Window Interface Browser

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 the interface 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.

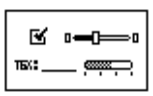
2. **Customize the base window's properties.**
   Double-click SELECT on the base window's interior or footer to open its property window. Then, change the values in the fields.

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

Note – The changes you make on a property 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 (e.g. text panes and control areas) directly on a base window or popup window. However, you *must* place control elements, such as buttons and sliders, on a control area. You can place more than one control area in a window.



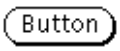
1. **Create a control area by dragging the Control Area glyph onto the base window.**
   Place and resize the control area so it looks like the area in the top of the base window in Figure 3-3 on page 29. This control area will contain the application's Format and Text Size buttons.

2. Customize the control area's properties.

Object	Property Field	Value
Control area (controls1)	Object Name	**top_controls**

Adding Buttons to the top_controls Control Area

You create two different kinds of buttons for this sample application: an abbreviated menu button and a button to display a popup 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.

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 (button1)	Object Name	**menu_button**
Left button (button1)	Label	**Format**
Left button (button1)	Type	Abbreviated Menu
Right button (button2)	Object Name	**popup_button**
Right button (button2)	Label	**Text Size...**

Later on, you will create a menu and attach it to the left button. Figure 3-3 shows how the interface should appear when you have completed these steps.

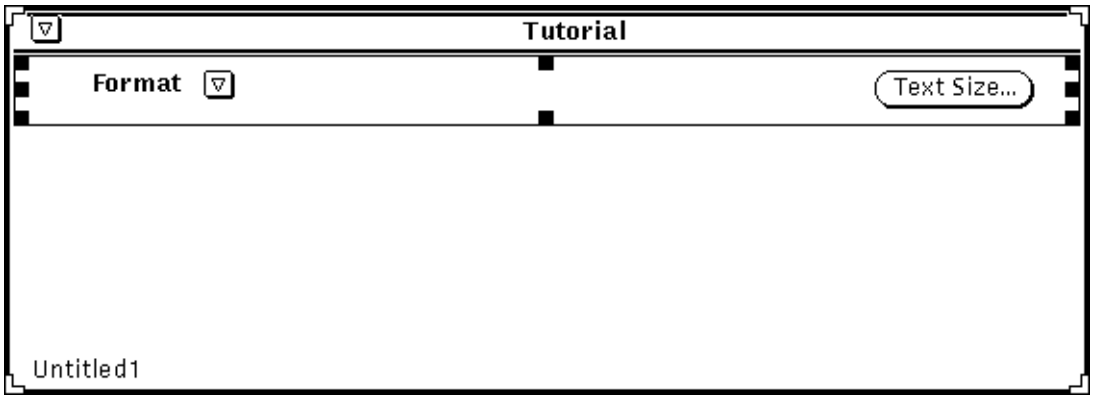


Figure 3-3 Base Window Interface with Buttons

Adding a Text Pane to the Tutorial Base Window

Devguide provides three objects that you can display and write text in: text fields, multiline text fields, and text panes.

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

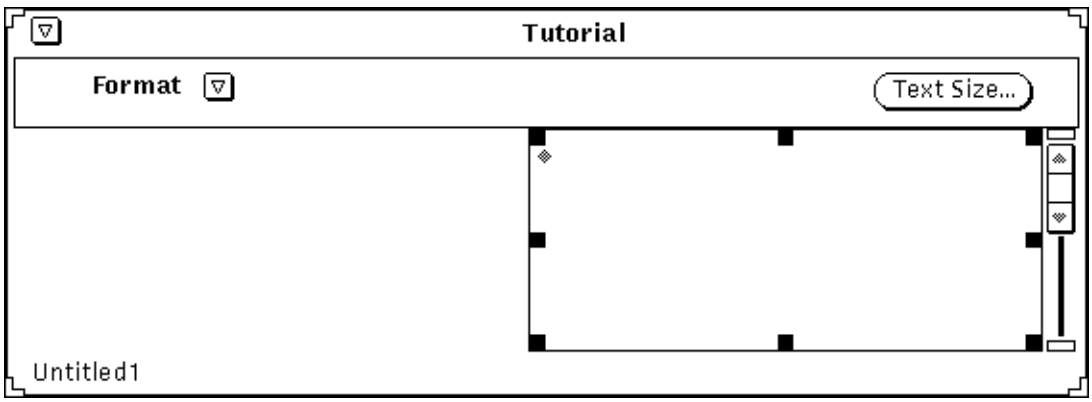


Figure 3-4 Base Window Interface with Text Pane

Creating Layered Control Areas

Devguide allows you to add layered panes to your application. You can layer control areas and canvas, text, and term panes. In the completed application, only one pane will be visible at a time. The application must include a connection—to make the pane visible—for each pane that you layer. 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 31.
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`**

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

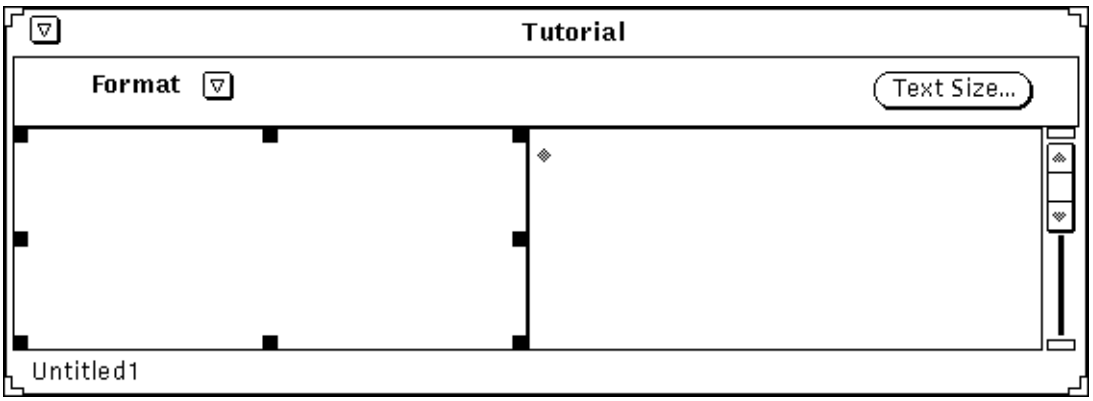


Figure 3-5 Base Window Interface with Layered Control Areas

Adding Controls to the Layered Control Areas

In this section you will add controls that control the text pane's font and style to the layered control areas. You will add an exclusive setting for the font to one control area and a checkbox setting for the style to the other.

To add an exclusive setting to the `font_controls` control area:

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

   Customize the setting's properties as follows:

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

2. **Customize the setting items' (Choices) properties.**
   The Choices scrolling list contains two choices.

Object	Property Field	Value
First Choice	Label	**Roman**
Second Choice	Label	**Lucida**

Devguide automatically sets the first choice as the default choice in the exclusive setting, unless you alter the choices' Not Selected/Selected property.

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 Devguide's 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:**
Checkbox Setting	Rows/Columns	(Rows) 1

3. **Customize the individual checkboxes' (Choices) properties.**
   The Choices scrolling list contains two choices.

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

Figure 3-6 shows the interface with `style_controls` in front; Figure 3-1 on page 24 shows it with `font_controls` in front.

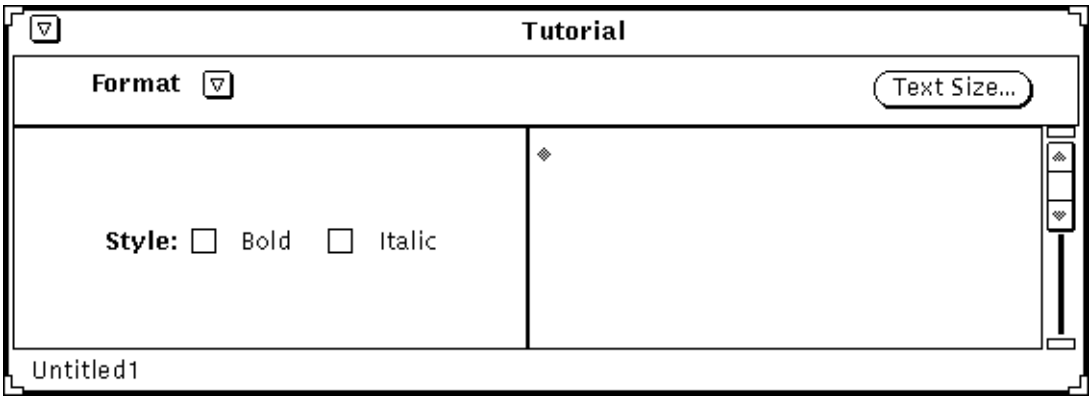


Figure 3-6 Base Window Interface with Style Controls Displayed

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.**
   Devguide appends the `.G` extension to the interface file.

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`. Customize the menu's properties.

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 of the tutorial, you create an additional interface consisting of a pop-up window, a control area, and slider. Figure 3-7 on page 35 shows the interface as it appears in the Browser.

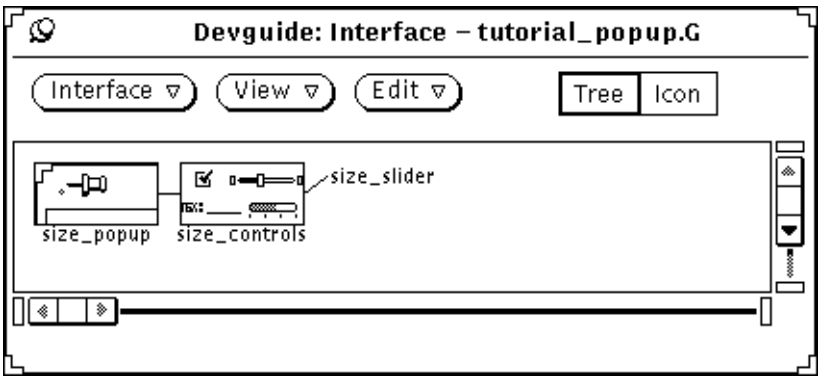
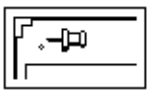


Figure 3-7 Pop-up Window Interface Browser

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	Window Parent	**`base_win`**
Pop-up window	Label	**`Text Size`**

Adding a Control Area to the Pop-up Window

Add this control area to the pop-up window the same way you added the control areas to the base window.

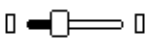
1. **Drag the Control Area glyph onto the pop-up window and resize the control area.**
   Resize the control area to fill the pop-up window.

2. **Customize the control area's properties.**

Object	Property Field	Value
Control area (controls1)	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.**
   Position the slider as shown in Figure 3-8 on page 37.

2. **Customize the slider's properties.**

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	Ticks	**13**
Slider	Tick String Min	**4**
Slider	Tick String Max	**28**
Slider	Initial Value	**12**

Figure 3-8 shows the completed pop-up window interface (after it has been saved).

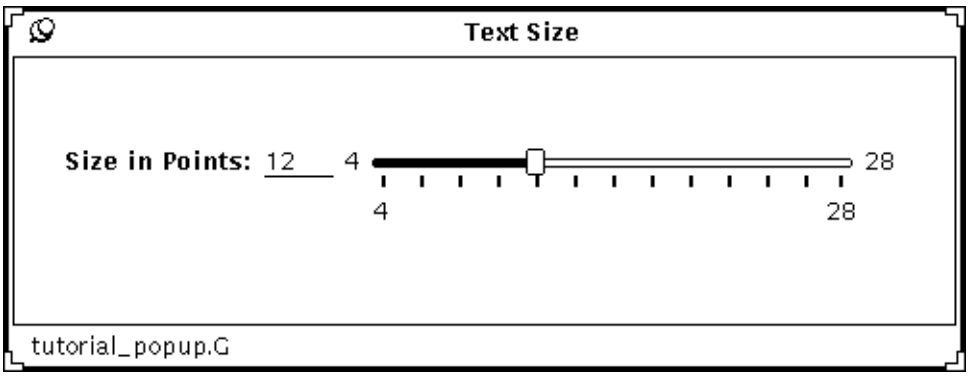


Figure 3-8 Completed Pop-Up Window Interface

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, you can create a project file. 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; in particular, icons 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`. The `tutorial` project consists of `tutorial_main.G` and `tutorial_popup.G`.

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 must insert custom code into

In this section you establish 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 is pressed.

You also set up 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. Then click SELECT on the Connect button to complete 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 entire 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 Devguide's Properties menu.
2. **Click SELECT on Connections... in the upper half of the Menu Editor window.**
   The Connections Manager window appears.
3. **Choose Buttons from the Connections Manager's 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.

You can also attach a menu to a button by using the Menu button on the Buttons property window. See the Reference chapter of the *OpenWindows Developer's Guide: User's Guide* for details.

Creating Connections to Switch between Layered Control Areas

To switch between the layered control areas, you need to create two connections: one for each menu item in the Format menu.

- Show `style_controls` when the user chooses Style.
- Show `font_controls` when the user chooses Font.

To create the connections (in the Connections 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.

Create the following connections

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

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

Use drag and link to create this connection. Select the Text Size... button. Press the Meta key and drag the pointer from the button to the pop-up window. The pointer turns into a cord with a plug at the end.

Note – Use the key marked "Alt" if there is no Meta key (usually marked as a diamond) on your keyboard.

When you release the SELECT button, the Connections Manager 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

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. Specify the connection as follows.

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

Creating Connections to Set Font and Style

To set up connections to set the text's font and style, choose Setting Items from the Source and Target menus. The item names displayed in the scrolling lists are prefixed with the setting name; for example, font.Lucida. Specify the connection as follows.

Source	Target	When	Action	Arg
Lucida	Lucida	Notify	CallFunction	**`set_lucida`**
Roman	Roman	Notify	CallFunction	**`set_roman`**
Bold	Bold	Notify	CallFunction	**`set_bold`**
Italic	Italic	Notify	CallFunction	**`set_italic`**

Saving

Save your application in order to save the connections.

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 behave. First choose Test mode. Then you can display the Format menu, scroll through the layered panes, and display the Text Size pop-up window.

1. **Choose Test mode by clicking SELECT on Test in Devguide's base window controls area.**
   The Test choice is highlighted. All the predefined connections in your application will work. The only connections that do not work are CallFunction and ExecuteCode connections.

2. **Put the pointer over the text pane you created and enter some sample text.**
   Notice that the scroll bar is activated if you enter more text than can appear in the text pane at one time. *The text you enter is not saved*. The text pane will be empty when you start up the completed tutorial. *The font of the text you enter in Test mode will be Devguide's default font*.

3. **Click 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.

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 font size in Test mode.

7. **Click SELECT on Build to reenter Build mode.**

8. **Unload the previously-saved base window and pop-up window interface files.**

Generating Interface Source Code

1. **Use the Code Generator Tool or run the `gxv` command on your project file.**
   Invoke `gxv` with the `-p` option because `tutorial` is a project file. GXV lists the files it is currently reading and writing:

```
% gxv -p tutorial
 gxv: reading tutorial.P
 gxv: reading tutorial_basewindow.G
 gxv: writing tutorial_basewindow_ui.h
 gxv: writing tutorial_basewindow_ui.c
 gxv: writing tutorial_basewindow_stubs.c
 gxv: writing tutorial_basewindow.info
 gxv: reading tutorial_popup.G
 gxv: writing tutorial_popup_ui.h
 gxv: writing tutorial_popup_ui.c
 gxv: writing tutorial_popup_stubs.c
 gxv: writing tutorial_popup.info
 gxv: writing tutorial.c
 gxv: writing tutorial.h
 gxv: writing Makefile
```

Creating Custom Code Using GXV Files

In this section you alter four GXV-generated files to make the CallFunction connections work. The callback functions GXV generates for these connections contain print statements that you replace with custom ANSI C code. You also need to define global variables to register the user's choice of Bold and/or Italic, and Roman or Lucida.

You insert a common function called `change_font()` inside the `set_roman()`, `set_lucida()`, `set_italic()`, `set_bold()`, and `set_textsize()` callbacks to change the text's font, style, and size.

Open the files in your favorite text editor and save all changes.

Customizing the `tutorial.c` *File*

1. **Insert these global variables into the file before the definition of** `main()`**.** The text in the text pane when you first run the application is twelve point Roman regular font.

```
Attr_attribute INSTANCE;
int Bold = 0;
int Italic = 0;
int Lucida = 0;
int Size = 12;

/*
 * main for project tutorial
*/
void
main(int argc,char **argv)
```

2. **Insert a call to** `change_font()` **right before the call to** `xv_main_loop()` **inside** `main()`**.**
   This initializes the text's style and font to the values set above. `change_font()` must be called after all UI objects are initialized and before XView polls for events in `xv_main_loop()`.

```
            change_font();

            /*
            * Turn control over to XView.
            */

xv_main_loop(tutorial_main_base_win->base_win);
```

Customizing the `tutorial.h` *File*

1. **Add extern declarations and define the constants TRUE and FALSE.**
   Include the extern declarations so both `_stubs.c` files can access the variables and `change_font()` function. Redefine 1 and 0 as TRUE and FALSE, respectively, to make the code more readable and boolean variables more obvious.

```
#define TRUE 1
#define FALSE 0

extern int Bold;
extern int Italic;
extern int Lucida;
extern int Size;

extern void change_font();
```

Customizing the `tutorial_popup_stubs.c` *File*

1. **Replace the** `fprintf()` **statement in** `set_textsize()` **with code to read the slider value and scale the font.**
   First retrieve the user's choice for text size, and then set the font size accordingly.

```
Size = value;
change_font();
```

Customizing the `tutorial_main_stubs.c` *File*

You created CallFunction connections for both checkbox and exclusive settings earlier in this tutorial. GXV generates callbacks for each individual item connection you create, as well as a wrapper function that executes item callbacks as appropriate.

For example, in this tutorial GXV generates `tutorial_main_base_win_font_notify_callback()`, which calls `set_roman()` and `set_lucida()`. (GXV constructs names for callback wrapper functions by concatenating together the names of certain objects with the name of the interface containing the function.)

The name of the callback that calls `set_bold` and `set_italic` is given in Step 4.

1. **Add the following line before the statement** `#include "tutorial.h"`**.**
   This provides access to XView's font library.

```
#include <xview/xv_xrect.h>
#include <xview/font.h>
#include "tutorial.h"
```

2. **Replace the call to** `fputs()` **in** `set_roman()` **with code to set the font to Roman.**
   First record the font choice as Roman and then set the font accordingly.

```
Lucida = FALSE;
change_font();
```

3. **Replace the call to `fputs()` in `set_lucida()` with code to set the font to Lucida.**
   First record the font choice as Lucida and then set the font accordingly.

```
Lucida = TRUE;
change_font();
```

4. **Insert the following right before the `for` loop in `tutorial_main_base_win_style_notify_callback`.**
   Initially clear the Bold and Italic values; function calls within the `for` loop reset these booleans to current user input values.

```
Bold = Italic = FALSE;
```

5. **Replace the call to `fputs()` in `set_bold()` with code to set the font type to bold.**
   First record the text style choice as Bold and then set the style accordingly.

```
Bold = TRUE;
change_font();
```

6. **Replace the call to `fputs()` in `set_italic()` with code to set the font type to italic.**
   First record the text style choice as Italic and then set the style accordingly.

```
Italic = TRUE;
change_font();
```

7. **Insert the code for** `change_font()` **at the end of the** `tutorial_main_stubs.c` **file.**

```
void
change_font()
{
        Xv_opaque  text_window;
        Xv_opaque  base_window;
        Xv_opaque  font;
        char       *style;

        text_window =
        Tutorial_main_base_win -> textpane1;

        base_window =
        Tutorial_main_base_win -> base_win;

        if (Bold && Italic)
                    style = FONT_STYLE_BOLD_ITALIC;
        else if (Bold)
                    style = FONT_STYLE_BOLD;
        else if (Italic)
                    style = FONT_STYLE_ITALIC;
        else
                    style = FONT_STYLE_NORMAL;

        /* Set the window busy; finding the font the first time can take a
                    while */
        xv_set(base_window, FRAME_BUSY, TRUE, NULL);

        /* Find the font */
        font = xv_find(text_window, FONT,
                    FONT_FAMILY,
                    (Lucida ? FONT_FAMILY_LUCIDA : FONT_FAMILY_ROMAN),
                    FONT_STYLE, style,
                    FONT_SIZE, Size,
                    NULL);

        xv_set(text_window, WIN_FONT, font, NULL);
        xv_set(base_window, FRAME_BUSY, FALSE, NULL);
}
```

3

Compiling Code

To compile code for your sample application:

- Use the Code Generator Tool, or
- Enter `make` in the shell tool. `make` references the GXV-generated `Makefile` as it compiles your code.

If you build an application containing any `.c` files that aren't generated by Devguide, you should include the files' names in the `SOURCES.C` line in the `Makefile`. This tutorial application only contains Devguide-generated files.

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

```
% make
```

   An executable file named `tutorial` is produced, assuming you've made no errors.

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

```
% tutorial
```

You now have a working application. The base window appears, with the Format menu button, Text Size... button, Font exclusive setting, and text pane visible. You can enter text into the text pane. Try changing the font, style, and size of the text.

This completes the tutorial. The remainder of this programming guide discusses generating GXV code, describes GXV's libraries, and provides some programming tips.