Chapter 6 — Form Designer Project

In this chapter, we will discuss how to build the Form Designer project that appears in Chapter 13 of *Smalltalk Programming for Windows.* By now, we figured that you'd be getting a little tired of simply converting applications from direct Smalltalk/V implementation to using WindowBuilder Pro, so we're taking a different approach.

To extend the functionality of the Form Design application and to make it more reusable, as well as to show off some of the power of WindowBuilder Pro CompositePane components, we created a special CompositePane for this application and then rebuilt the application. This chapter, then, gives you more exposure to the total application development process as it might be approached in WindowBuilder Pro than earlier chapters, which have assumed the code for an application had already magically appeared or at least been created outside the context of the WindowBuilder Pro design.

We'll begin by looking at the EntryFieldGroup CompositePane object and how it works. Then we'll incorporate such an object into a window that forms the basis of our FormDemoWB application. For the first time in this manual, the new application will not be a subclass of the corresponding class in *Smalltalk Programming for Windows,* since we are changing its design so radically. As a result, we will spend more time talking about what it takes to finish the application than we have in previous chapters.

The EntryFieldGroup CompositePane

If you've read Chapter 13 of *Smalltalk Programming for Windows,* you will recall that its purpose was to create a general-purpose, reusable design that would permit you to define a form with a minimum amount of effort. To do so, it defined a new class, FormPane, as a subclass of GroupPane. To build an application with this tool, you would define a separate FormPane instance for each field on a form.

Taking advantage of the concept of the CompositePane in WindowBuilder Pro, we extend that concept logically to design an EntryFieldGroup.

There are some important differences between the fields created in the FormPane method `openWithLabels:withValues:formatted:` in *Smalltalk Programming for Windows.* In addition to the obvious fact that an EntryFieldGroup defines an entire set of fields in one object, EntryFieldGroup objects do not include the inherent capability to define a

format for the strings because they use EntryFields rather than FormattedEntryFields. This ability can be easily added by subclassing EntryFieldGroup and overriding the `fieldClass` method to return a different type of EntryField (such as FormattedEntryField or our own EnhancedEntryField)

NOTE

> If you want to run the Form Designer application in *Smalltalk Programming for Windows*, don't forget that you must file in the FormattedEntryField class from the Examples folder that comes with Smalltalk/V. Information about how to do this and what this class adds to your Smalltalk image's capabilities is on page 260 of the book.

Let's take a look at what an EntryFieldGroup compositepane includes and how it behaves. Open a new window in WindowBuilder Pro and select the CompositePane tool from the tool palette. (It's the last icon on the left toolbar.) Now load the cursor with an EntryFieldGroup pane by clicking on the third icon below the arrow of the right tool palette. Place it arbitrarily; we don't intend to do anything serious with it, just examine its construction and behavior.

Double-click on the EntryFieldGroup object. A dialog like the one shown in Figure 6-1 appears.

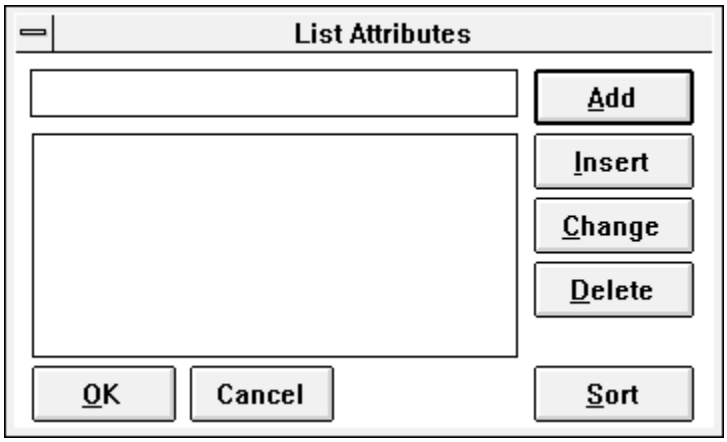

Figure 6-1. EntryFieldGroup List Editing Dialog

In many ways, this dialog is quite similar to those you've seen as you've built radiobuttongroups and listboxes in WindowBuilder Pro. You simply provide WindowBuilder Pro with a list of the labels you want to create in

your form design by typing them and pressing Return. You can arrange them in any order you want (including sorting) before clicking the **OK** button. For now, just press the **Cancel** button in this dialog.

When you create forms for the kind of application we're building here, you probably don't want a labeled group box around the fields (though you may well want them if you were building this set of form entry fields as part of a larger window or dialog). In the **Style** popup menu, pick the "noGroupBox" style.

NOTE

> You should notice that an EntryFieldGroup compositepane has a style called "verticalScrollbar" that will allow you to create a scrolling group of entry fields. If you've ever tried to place several related controls into a scrolling object in a window or dialog without the help of WindowBuilder Pro, you can appreciate how useful this capability is. It is not confined to EntryFieldGroups, either.

Let's examine the code behind an EntryFieldGroup object. You can do this by switching out of WindowBuilder Pro for a moment and opening a Smalltalk/V Class Hierarchy Browser on the EntryFieldGroup class. When you do so, you'll notice that its instance variables are called `label` and `dictionary`. The latter is the key to the behavior of an EntryFieldGroup object, so we'll spend some time looking at it.

An EntryFieldGroup has a Dictionary of Strings as its content. The keys in this Dictionary are the labels on the fields; the values in the Dictionary are the contents of the entry fields that have been supplied by the user.

When an EntryFieldGroup is created, it dynamically creates a set of right-aligned StaticText widgets and corresponding EntryFields.

Look at the EntryFieldGroup's instance method called `contents:` to see how the Dictionary is structured and managed. To get a concrete example of this process, browse the class FormDemoWB's `open` method. Notice the following line in the portion of the open method that creates the EntryFieldGroup:

```
contents: #('Name' 'Address' 'City' 'State' 'Zip'
'Phone #');
```

As you can see, when you add a list of labels to the EntryFieldGroup, WindowBuilder Pro builds an array of strings consisting of one string for each label you enter. The `contents:` method of the EntryFieldGroup class then converts this array into a Dictionary.

Constructing the Demo Application in WindowBuilder Pro

With this understanding of how the EntryFieldGroup class works, we are ready to build this new version of the form designer application. As you'll see, it is relatively trivial once we have a CompositePane object with the power and versatility of the EntryFieldGroup.

The application will look something like Figure 6-2 when we finish it. (Note that there are some cosmetic differences between this design and the one in *Smalltalk Programming for Windows,* on p. 256. Different designers have different ideas about where buttons should be located and how they should look. That's what makes a circus!)

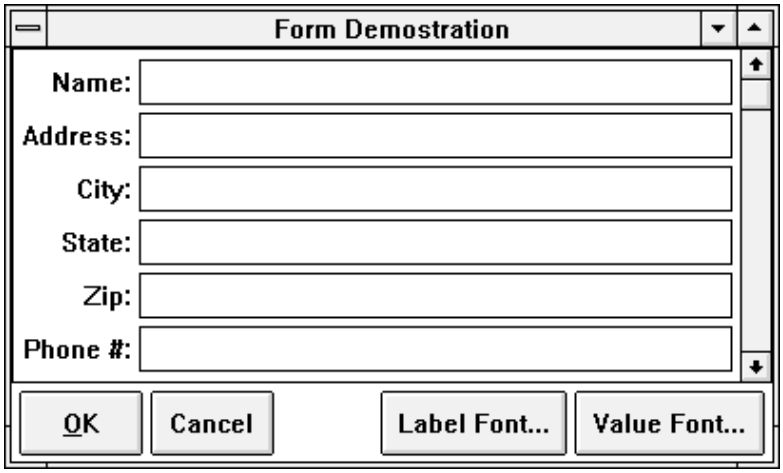

Figure 6-2. Finished Form Demo Application

Follow these steps to create the FormDemoWB application:

1. Open WindowBuilder Pro to edit a new window.
2. Resize the window to be large enough to accommodate the six fields that will appear in the EntryFieldGroup. We found a size of 408 x 244 worked well.
3. Load your cursor with an EntryFieldGroup object.
4. Drag a new EntryFieldGroup object so that it occupies most of the top portion of the client area of your window. (A size of 402 x 178 worked well for our design.)
5. From the **Style** menu, choose "verticalScrollbar." (This will allow the user to resize the window without losing the ability to access any of the fields or their values.)

6. Name this pane “form” by typing that name into the **Name:** field of the WindowBuilder Pro window editor.
7. Double-click the EntryFieldGroup widget. In the ensuing dialog, enter the labels “Name”, “Address”, “City”, “State”, “Zip”, and “Phone #” in that order. (Note that you need not include the colons following each label; an EntryFieldGroup automatically appends a colon to each label when it creates the widget.)
8. Click the **OK** button to store these values. When the window reappears, notice that it now has the six labeled fields correctly placed for you. (This just saved you a great deal of work compared to the approach we had to use in *Smalltalk Programming for Windows* without the support of WindowBuilder Pro! The `openWithLabels: withValues:formatted:` method and the `defineFields: defaultValues:formatted:` method, as well as supporting methods such as `nextLabelOrigin:of:widest:` and `nextValueOrigin:of:widest:` have been completely replaced by this single compositepane.)
9. Load your cursor with a PushButton object.
10. Create and place two buttons in the lower right portion of the window. Name them Label Font..., and Value Font... Align and distribute them to your liking.
11. Connect each of these buttons’ `clicked` method. The “Label Font...” button will activate the `labelFont` method when it is clicked. Similarly, the “Value Font...” button will activate the `valueFont` method when clicked.
12. Load your cursor with an OKCancelPane composite pane object. You can get this object from the **Add | Composite** submenu or from the CompositePane tool palette, where it is the fifth icon below the selector arrow (if you have installed the CompositePane examples from the distribution disk).
13. Place this compositepane in the lower left corner of the window.
14. Connect the `cancel` event for the OKCancelPane to a method called `cancel` and the `ok` event to a method called `ok.`

NOTE

Even though the methods themselves are called `ok:` and `cancel:` because they require the pane to be passed in as an argument, you don’t need to supply the colon in naming the methods in WindowBuilder Pro. In fact, if you do supply them, WindowBuilder Pro simply removes them for you!

15. Test the window. Click on the “Value Font...” button. You should see a dialog like the one in Figure 6-3. Even though WindowBuilder Pro saves your window (i.e., the Smalltalk code necessary to create and manage it) when you test an unsaved window, it does not do so in a ViewManager subclass. The functionality of such a window is limited, as this dialog informs you.

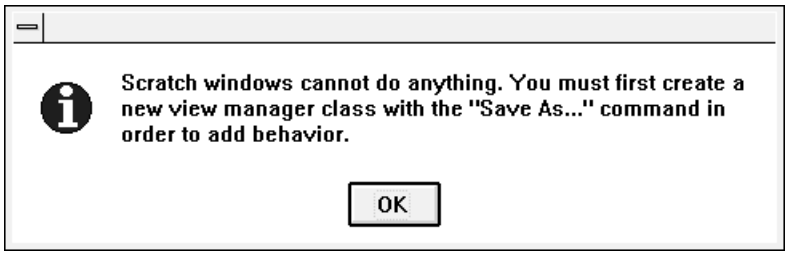

Figure 6-3. Dialog for Scratch Window Test

16. Save the window. Make it a subclass of ViewManager.
17. Test the window again. Everything works, but the buttons are not hooked up. That is part of finishing the application, which is our next topic.

Finishing the Application

Open a Class Hierarchy Browser on your newly created application. Notice that, as expected, WindowBuilder Pro has created empty methods (sometimes called “stubs”) for `ok:`, `cancel:`, `labelFont:`, and `valueFont:`. Finishing the application consists of fleshing out these blank methods.

The “ok:” Method

The `ok:` method in our version of the Form Demo application is a little more complex than the one in *Smalltalk Programming for Windows* because of the fact that the values we want to display in the Transcript are stored in a Dictionary associated with the EntryFieldGroup widget in our window. Here is the code for the `ok:` method:

```
ok: aPane
    "Callback for the #ok event in an unnamed
     OkCancelPane. (Generated by WindowBuilder)"
    | fieldValues |
    Transcript cr; show: 'The responses are: '.
        (fieldValues := (self paneNamed: ‘form’)
            contents) keysDo: [ :key |
            Transcript cr; show:
                key,' ==> ',
                (fieldValues at: key)].
    self close.
```

We assign the contents of the EntryFieldGroup object to a local variable called `fieldValues`, then go through the Dictionary and display each entry’s key and value. When the display is complete, we close the window.

The “cancel:” Method

The `cancel:` method only needs to close the window, so its code is quite simple:

```
cancel: aPane
    "Callback for the #cancel event in an unnamed
     OkCancelPane." (Generated by WindowBuilder)”

    self close
```

The Font-Changing Methods

The two methods for changing fonts in the Form Demo application’s window are quite similar. The first is `labelFont:` and is used to change the font with which the prompts or labels in the window are displayed. Here is its code:

```
labelFont: aPane
    "Callback for the #clicked event in an unnamed
     Button (contents is ‘Label Font...’).
     (Generated by WindowBuilder)"

    (self paneNamed: 'form') setLabelFont:
        (Font chooseAFont: 'Choose a new Field Font')
```

This code is somewhat simpler than that shown in *Smalltalk Programming*

for Windows at pages 270-271. The reason for this simplicity is found in the fact that the EntryFormGroup class implements a method called `setLabelFont:`, which is much closer in its content to the code in the book:

```
setLabelFont: aFont
    (self children select: [:pane |
       pane class == StaticText ]) do: [:field |
       field font: aFont].
     self update
```

The case is identical with the `valueFont:` method:

```
valueFont: aPane
    "Callback for the #clicked event in an unnamed
     Button (contents is 'Value Font...').
     (Generated by WindowBuilder)"

    (self paneNamed: 'form') setValueFont:
       (Font chooseAFont: 'Choose a new Field Font')
```

Its associated `setValueFont:` method in class EntryFieldGroup looks almost identical to the `setLabelFont:` method, so we won't take up space, paper, trees, and your valuable time by reproducing it here.

Testing the Final Application

Once you've fleshed out these four methods, your application is finished. You can test it either by launching it from the Transcript or a workspace window or by opening the WindowBuilder Pro editor on it.

It is instructive and important for you to note that it is in fact possible to open this application in WindowBuilder Pro even though you've now added some functionality to it that was not part of what WindowBuilder Pro supplied. The only method that WindowBuilder Pro generated (and therefore feels like it "owns") is the `createViews` (or `open`) method.

Enter some values into the fields and, when you press the **OK** button, notice that your entries are reproduced in the Smalltalk Transcript.

Some Closing Thoughts

You have now had enough experience with WindowBuilder Pro that you should be fairly comfortable with what it can do and how to find your way around in it. You have not scratched the surface of some of the more intriguing options in WindowBuilder Pro, however. All of these are covered in the Reference Guide, which you should peruse as you find the need to do things that we have not explicitly discussed here.

As you create new Smalltalk/V applications using WindowBuilder Pro, try to remain aware of the possibilities of creating new custom and compositepanes that will make subsequent projects even easier to build. Your goal should be to get to the point where you have to create little or no new code for each application. Maximize reuse of code!

Happy Window Building!