Chapter 4 — The Calendar Application

In this chapter, we will take a look at how you might go about building the Calendar application in *Smalltalk Programming for Windows.* That application is created in three stages in the book, so in this chapter, we will examine each phase of the development. At each phase, we will look first at how you might use WindowBuilder Pro to create any new user interface elements. Then we'll take a look at how the code automatically generated by WindowBuilder Pro differs from the manually created code in the book and what the impact of those differences is on the coding process. Finally, we'll describe how to complete each specific stage of the application by looking at the code you would write to supplement that generated by WindowBuilder Pro.

You might want to take time to review Chapters 6-9 and 14-15 of *Smalltalk Programming for Windows* before you read the rest of this chapter because we assume you are familiar with the application we're building and with how it was done originally without the aid of WindowBuilder Pro.

NOTE

All of the source code for the WindowBuilder Pro versions of this application and those in the remaining chapters of this manual is available on the disk accompanying this manual. You need not type the code manually into your image unless you learn better that way.

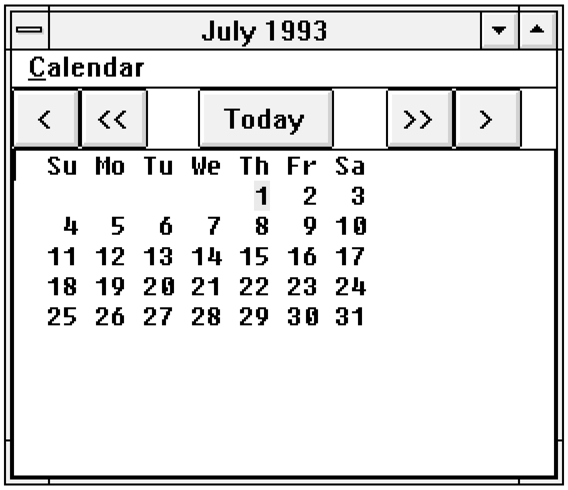
The Application: Phase One

In Phase One of this project, we create a simple but functional calendar. This single-window application includes a set of related menus.

To prepare for this exploration, file in the code for Chapter 7 from the disk accompanying *Smalltalk Programming for Windows* (or from the CHAP4PH1 subdirectory created when you installed WindowBuilder Pro) so that you have Calendar and CalendarApp classes defined in your image. You may wish to save the image when you have done so. Since this code employs an `open` method rather than a `createViews` method, you should enable the "WindowBuilder 2.0 Code Generation" Add-In for the remainder of the tutorial.

Constructing the Interface in WindowBuilder Pro

Figure 4-1 shows what the finished Calendar application will look like after we complete Phase One of its development in WindowBuilder Pro.



```
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

FIgure 4-1. First Phase Window Design

Although you can go about constructing a window in WindowBuilder Pro in almost any order, we'll modify the window with which WindowBuilder Pro starts the session, then add the buttons and the text pane, and finally we'll create the menu for the window. Here are the steps you should follow.

1. Launch WindowBuilder Pro by choosing "New Window" from the WindowBuilder Pro menu in the Smalltalk/V transcript.
2. Resize the window with which WindowBuilder Pro starts you so that it is approximately 280 x 240. You can do this by dragging the lower right corner around as you watch the dimensions change in the small button at the lower right of the editing area or by clicking on that small button and providing the size explicitly in the resulting dialog.
3. Select the Push Button tool.
4. Create a small button in the upper left corner of the window. (Don't worry about leaving room for the menubar; WindowBuilder Pro will handle this problem for you as you'll see shortly.) A size of 34 x 29 seems to work well for the size window we've created.
5. Change the name of this button to "<" (a shifted comma on the standard PC keyboard).
6. While it's still selected, choose **Edit | Copy** or type Control-C to copy the button to the Clipboard.
7. Select **Edit | Paste** or type Control-V to paste the button from the clipboard to your cursor.

8. Using the *right* mouse button, click immediately to the right of the button you just created. This creates a new button of the same size and name while leaving your cursor loaded with the button object for subsequent quick pastes.
9. Change the name of the newly placed button to “<<”.
10. Repeat Step 8 two more times, renaming the newly placed buttons to “>” and “>>” and placing them appropriately.
11. Create another new pushbutton near the top center of the window. You can do this by pasting the button with which your cursor is now loaded or you can create a new button from scratch. Either way, you’ll have to resize it to get it right.
12. Name the new button “Today.”
13. Click on the button closest to the top of the window and then Shift-click on all the others until all five buttons are selected.
14. Click the **Align Tops** icon on the toolbar or choose **Align | Top** (or press its keyboard shortcut, Shift-Control-T) to align the buttons along the top. If you created the **Today** button manually, you may have to shrink it vertically. Do this by clicking on any of the other buttons (since they’re all the same height), Shift-clicking the **Today** button, and then clicking on the **Replicate Height** icon on the toolbar. You may also select **Size | Replicate Height** (or use the keyboard equivalent, Control-Shift-H).
15. Your window so far should look like Figure 4-2.

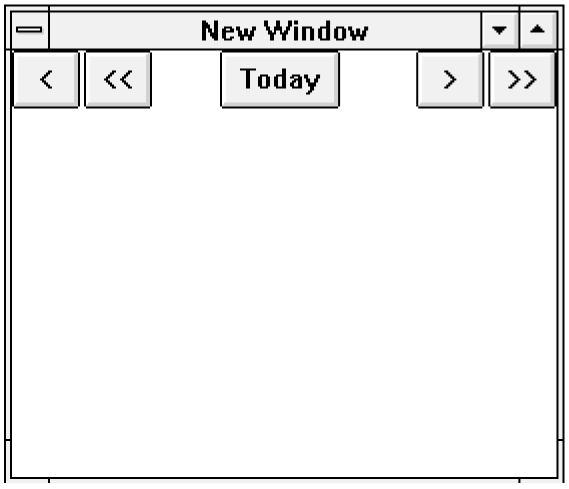


Figure 4-2. Buttons Placed in CalendarAppWB Window

16. Now let's connect each button to its appropriate method name. Select the first button on the left ("<"). Be sure that "clicked" is showing in the **When** editing area, and type `monthBack` into the **Perform** editing area.

NOTE

If you are using a copy of *Smalltalk Programming for Windows* from the First Printing, you will find the arrow-like buttons in the book are switched. Those that point back are tied to methods that look ahead and vice versa. We'll follow the correct placement in this discussion.

17. Select each of the other buttons in turn and set their **Perform** methods to, respectively, `yearBack`, `today`, `monthAhead`, and `yearAhead`. (Notice that as you provide names of methods to be performed when the chosen events occur, the event names change to put an asterisk in front of them. This is WindowBuilder Pro's way of letting you know you've connected that event to a method.)
18. Select the TextPane tool from the toolbar. (See Figure 4-3).

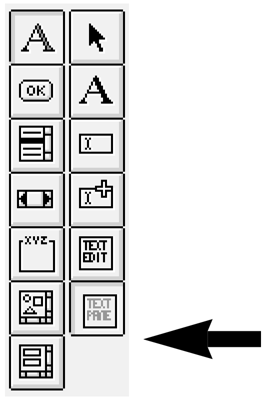


Figure 4-3. The TextPane Tool

19. Drag a TextPane object so that it starts at the left edge of the window and immediately below the row of buttons and stretches to the lower right corner of the window.
20. From the **Style** popup menu in the property editing area below the window region of WindowBuilder Pro, select `noScrollBars`.
21. Click in the window itself and select the `activate` event from the **When** popup. Then type `activate` for the **Perform** entry.

22. This is a good time to save your work. **Select File | Save** or **File | Save As...** and, in the resulting dialog, select CalendarApp as the superclass of the window you're about to save.
23. Give your window the name CalendarAppWB (or, frankly, any other name you like) and click **OK** or press Return.
24. We're ready for the final interface step, adding the menu to our window. Make sure the window is selected. You can do this by clicking anywhere in the editing area outside the window or by clicking on the window's title bar.
25. Click on the little menu-shaped button beneath the bottom right of the window editing area. A dialog box like the one shown in Figure 4-4 appears.

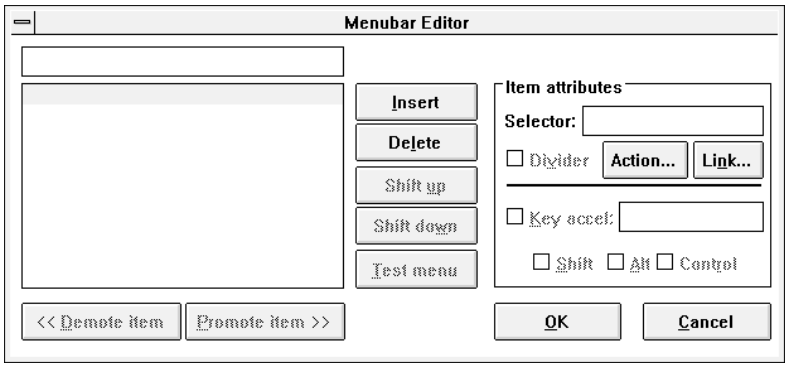


Figure 4-4. Empty Menu Editing Dialog

26. Type "&Calendar"(or "~Calendar" for OS/2) into the editing rectangle in the menu editor and press Return. The word "&Calendar" becomes the first one in the list pane appearing below the text editing rectangle in the dialog.

NOTE

The ampersand ("&") is a symbol that tells Windows to underline the next character in the menu or menu item's name and to use it as an accelerator key. The user can activate this menu choice from the keyboard. OS/2 uses the tilde ("~") for this purpose.

27. Type the rest of the menu entries as follows, pressing return or clicking on the **Insert** button after each: &Look, &Month Ahead, &Year Ahead, — (a hyphen indicates a divider), Month &Back, Year Back, &Go To Today, Go to Month and Year, — , &Show Special Days, &Refresh.

28. Now we need to arrange these menu items into an appropriate hierarchy. Start at the top of the menu items by clicking on the &Look entry. Notice that the **Promote Item** button is highlighted. Click on it.
29. The &Look entry is now shifted to the right and is preceded by some small dots indicating it's been indented in the menu structure. Select the next entry down the list, &Month Ahead. Click on the **Promote Item** button twice to indent it under the &Look item. Do the same for the next four items in the menu.
30. Promote the next five items once each.
31. Go back and select the &Month Ahead item. Notice that the right half of the Menubar Editor allows you to set attributes for selected menu items.
32. We want to associate the user's selecting the Month Ahead item with the method called `monthAhead`. Type that method name into the **Selector** editing rectangle.
33. Now associate the other menu items with their related selectors as follows:

    Year Ahead with `yearAhead`

    Month Back with `monthBack`

    Year Back with `yearBack`

    Go To Today with `today`

    Show Special Days with `activate`

    Refresh with `displayCalendar`

    (All of these are self-evident except the last two. The displaying of special days takes place whenever the calendar is updated, which takes place when it receives an activate message. Displaying the calendar will refresh its contents, as is the case with any Smalltalk window.)
34. As soon as you have finished defining the menu, click the **OK** button in the Menubar Editor. When you return to the window editing environment, you'll see that WindowBuilder Pro has added the menu to the window's menubar and properly re-positioned the buttons.
35. Test the window and open the menu to convince yourself the menu items are really there. You can even activate some of the menu choices if you like, although the results may be unpredictable depending on whether you have previously played with the Calendar application and created instances, for example.

36. The last thing we have to do is get the TextPane to respond to some events. Select it and give it the name calendarPane.
37. In the event-editing portion of the window (lower right corner), choose “getContents” and type `displayCalendar:` as the method to be performed when this pane receives a `getContents` message.
38. Similarly, connect the method `calendarMenu:` with the TextPane’s `getPopupMenu` message.
39. Now let’s connect the window’s response to two messages: select the `activate` message from the **When** popup and type `activate` as the method to execute in response. Similarly, connect the `close` event with the `close` method.
40. Save your work.
41. Test your window.

Reviewing the Code

When you save your work in WindowBuilder Pro, it creates appropriate Smalltalk/V code to create the window you’ve designed and to set up the event processing as you’ve described it.

Compare the `open` method generated by WindowBuilder Pro (assuming the “WindowBuilder 2.0 Code Generation” Add-In has been enabled) and the `open` method in *Smalltalk Programming for Windows* on pages 133 - 135. Though there are a number of minor differences, focus on the following points:

- The menu is integrated into the WindowBuilder Pro `open` method, greatly simplifying the code over what appears in the book’s source code.
- Whereas the book explicitly names TopPane as the class of which the view is an instance, WindowBuilder Pro uses the less direct but more reusable approach of retrieving the name of the appropriate class from the method `topPaneClass` and then instantiates that object. This kind of generic reusability is something WindowBuilder Pro strives for throughout its design.
- Framing blocks and parameters are handled automatically for you in WindowBuilder Pro, which is arguably one of the product’s most important productivity contributions. However, the code produced by WindowBuilder Pro to handle this task is fairly obtuse-looking. **Appendix C** contains a detailed explanation of this code. If you are terminally curious, you can check it out now.

- The final line of the open method as it appears on p. 135 of the book is `self setUp`. This line does not appear in the method WindowBuilder Pro generates. This is because a cardinal rule in working with WindowBuilder Pro is that you should never directly modify the `open` (or `createViews`) method generated by the program. Since we don't run into the `setUp` message while we are constructing the UI for our application, Smalltalk doesn't know about the need for this step. To handle it in WindowBuilder Pro, we create a new method called `initWindow` which takes care of the pre-processing for us. Examine that method from the disk accompanying this book and you can see that it sends the `setUp` message as expected.

Finishing the Application

Given that we create CalendarAppWB as a subclass of CalendarApp, you can see that we rely on the application as we received it from *Smalltalk Programming for Windows* for most of the non-UI behavior. If that code were not available, we'd have to write all of the methods contained in that class. These methods are described in the book. If you were starting from scratch in this application with WindowBuilder Pro, you'd have to construct all of the methods in CalendarApp with the exceptions of `open` and `setUp`.

You may have noticed, as well, that WindowBuilder Pro generated a class method for CalendarAppWB called `wbCreated`. This method simply returns **true** when it is called. It is used by several methods in the Smalltalk/V image that need to adjust their behavior based on whether a particular window or dialog was created by WindowBuilder Pro.

Now that we've worked through the construction of the UI and the completion of the simplest version of the Calendar application, you might want to take a few minutes to examine the code as it appears in the class CalendarApp and as it appears in the class CalendarAppWB. Studying the differences, running the two versions intermittently, and experimenting with both will teach you a great deal about how WindowBuilder Pro works.

The Application: Phase Two

In Phase Two of this project, we add an appointment book to the calendar application we have just finished constructing. We add a second window to enter, edit, and track appointments. This work is described in Chapter 9 of *Smalltalk Programming for Windows,* which you may wish to review before proceeding.

Constructing the Second Window in WindowBuilder Pro

This is our first encounter with a multi-window application, so we need to see how to add and manage views in WindowBuilder Pro. On the other hand, since we've now had the experience of building several views in WindowBuilder Pro, we don't need to walk through the creation of this window in the same detail as we did the main view in the first phase of this project. Follow these steps:

1. If you are still in WindowBuilder Pro and editing the CalendarAppWB view, skip this step. If not, then from the WindowBuilder Pro menu in the Smalltalk/V Transcript window, choose **Edit Window...** and open the CalendarAppWB view.
2. From the **View** menu in the WindowBuilder Pro editing window, choose the **Create...** option. The ensuing dialog looks like Figure 4-5.

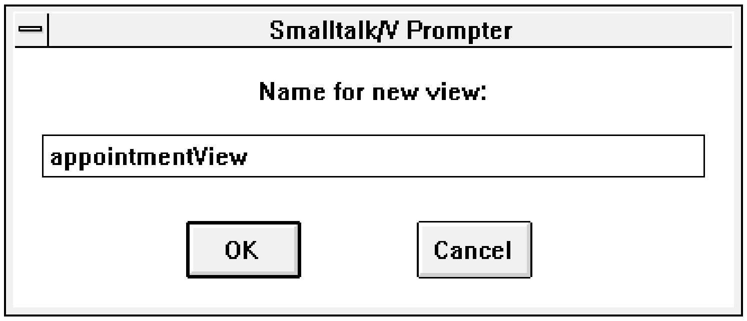


Figure 4-5. Dialog for Creating a New View

3. The present view in your application is called, by default, "mainView." Call this new view appointmentView and either press Return or click the **OK** button to create it. WindowBuilder Pro responds exactly as if you had launched it from the Transcript by creating a new window.
4. You can now create this view (a process we'll describe beginning with the next step). Any time you want to switch between multiple views in an application, select **View | Switch To...** and pick the view you want to work with next.
5. There is very little to the Appointment Book view we're calling appointmentView. It's essentially a moderately sized window with a single text pane and a menu. Resize the window given to you by WindowBuilder Pro to be something like 480 x 360.
6. Associate the `open` event for this window with the method called `open` and its `activate` event with the method of the same name.

7. Recall that in *Smalltalk Programming for Windows*, on page 176, we create a subclass of TextPane called NewTextPane that lets us handle mouse events in ways that are not accessible to us in Smalltalk's original TextPane class. Be sure you've filed that code into your image from the disk accompanying that book in your WindowBuilder Pro package before proceeding.
8. Add a NewTextPane component to the view, sizing it to occupy most of the space in the client area of the appointmentView window.
9. Connect the pane's `getContents` event with the method `displayCalendar`.
10. Now we want to move the menu that's currently associated with the mainView to the appointmentView and add a second menu to the menubar. Begin by copying the menu from mainView, an entry at a time, and pasting it into a newly created menu in appointmentView.
11. Once you have the menu copied to the appointmentView, just scroll to the end, and add a new menu called Appointments with two menu items, &Save (associated with the method `saveAppointment`) and S&how Schedule (connected to the method `showSchedule`).
12. Save your work.

You have now created a second view for your CalendarAppWB application.

Reviewing the Code

You won't find any revolutionary new differences between the code generated by WindowBuilder Pro for this phase of the Calendar application and that contained in *Smalltalk Programming for Windows* that we haven't already seen in our earlier code review. Notice that WindowBuilder Pro creates two topPaneClass objects now, one for the calendar window and one for the appointment book window.

Finishing the Application

We must take two more steps before we have completed our work with the second phase of the Calendar application in WindowBuilder Pro. First, we have to modify the `initWindow` method to enable us to refer to the various panes in our increasingly complex applications by their names, and second we must modify the `setUp` method to use these pane names.

In fact, it is not essential that we make these changes, but doing so has the virtue of making our code far more readable. Smalltalk/V does not include a built-in ability to refer to panes by their names, though many Smalltalk programmers find it necessary at some point in their experience to add this ability. WindowBuilder Pro provides it practically free; all you have to do is use the techniques described in this discussion to take advantage of this power.

Here is the code for the new `initWindow` method:

```
initWindow
    calendarPane := self paneNamed: 'calendar'.
    appointmentPane := self paneNamed: 'appointment'.
    calendarView := self mainView.
    appointmentView :=
        self viewNamed: 'appointmentView'.

    self setUp.
    self update.
    self updateViews: #all.
```

As you can see, this method simply defines names for the panes and views in the application. It leaves the Calendar view's name at mainView because that term is used throughout WindowBuilder Pro applications to refer to the primary view in an application.

The last three lines of the initWindow method are designed to deal with the issues of pane setup and initialization discussed in *Smalltalk Programming for Windows* on pages 192-194.

Now here is how we modify the code in the `setUp` method to be more readable and to match with the pane and view names we've just defined in the initWindow method:

```
setUp
    "setup the application"
    calendar := Calendar new pane: calendarPane.
    appointment :=
        AppointmentBook new pane: appointmentPane
    staticPane: (self paneNamed: 'holiday').
```

This is fairly straight-forward code. We now have completed our work on Phase Two of the Calendar application in WindowBuilder Pro.

The Application: Phase Three

In the third and final phase of this application, we add a view that is barely visible. This clock view (discussed in Chapter 15 of *Smalltalk*

Programming for Windows) displays the time continuously while the application is running and the clock is turned on. It demonstrates multiprocessing, which is an inherently non-visual component of an application. Still, we need an admittedly minimalist view to provide a place to update the time. And we need to make a few changes to other visual aspects of the application, notably the menus, to give the user a way to control some aspects of the clock's operation.

Creating the Clock View in WindowBuilder Pro

1. Open WindowBuilder Pro and edit the CalendarAppWB ViewManager object. Add a new view called clockView.
2. Size the window so that it shows only the title bar of the new window.
3. Change the title to "Clock."
4. Connect the `activate` and `close` events to methods of the same name.
5. Save the window.
6. From the **View** menu, choose **Switch To...** and then select the appointmentView from the resulting list.
7. Edit the menubar of this view to add a Clock menu with items called &Off (connected to the method `clockOff`, O&n (connected to `clockOn`) and &Add an Alarm (connected to no method).
8. Add two submenu items to the Add an Alarm menu item in the Clock menu. The first submenu is called &Daily and triggers the method `addDailyAlarm`. The second is called &For Current Date and triggers `addDateAlarm`.
9. Save your work.

Reviewing the Code

The code in the `open` method has changed only in ways that are predictable as a result of modifications we have made by adding a clockView and modifying the menu.

Finishing the Application

As was the case when we moved from Phase One to Phase Two of this project, we will modify both `initWindow` and `setUp` to take into account our new view. Here is the code for the `initWindow` method. (Added lines are in boldfaced type here but, of course, not in the Smalltalk/V Class Hierarchy Browser.)

```
initWindow

    calendarPane := self paneNamed: 'calendar'.
    appointmentPane := self paneNamed: 'appointment'.
    calendarView := self mainView.
    appointmentView :=
        self viewNamed: 'appointmentView'.
    clockView := self viewNamed: 'clockView'.

    clock := Clock new
        owner: self;
        when: #second perform: #updateClock:;
        when: #minute perform: #checkAlarms:;
        when: #hour perform: #hourlyChime:;
        when: #day  perform: #daysEnd:;
        start.

    self setUp.
    self update.
    self updateViews: #all.
```

As you would expect, we added a line to identify the new view with its name. Since this method's role is initialization, we also chose it as the place to define the events to which the clock will respond in the environment and how it will do so.

Now here's the code for the setUp method:

```
setUp
    "setup the application"
    calendar := Calendar new pane: calendarPane.
    appointment :=
        AppointmentBook new pane: appointmentPane
    staticPane: (self paneNamed: 'holiday').
    alarms := Dictionary new.
    self createAlarmsList.
```

The only setup processing specific to the clock object is the necessity of creating and initializing a Dictionary to hold alarms for the system.