Chapter 12
Working with Draw Objects

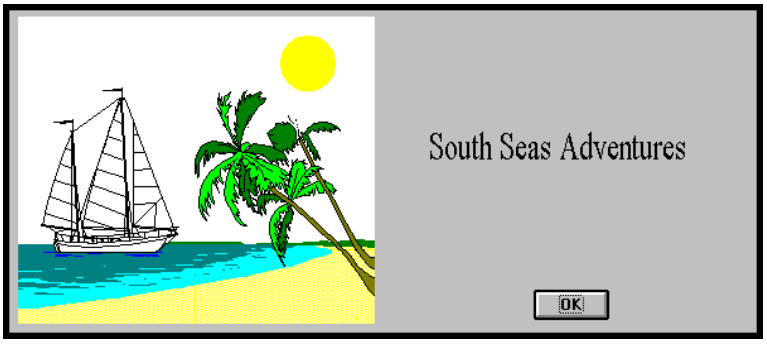
Objective

In this lesson, you will learn how to display and manipulate bitmap and text objects using the DrawObject classes. You will also see when it is appropriate to display these objects.

Overview

The opening dialog box of the South Seas Adventures application contains a bitmap and text, which were implemented using objects from the DrawObject hierarchy. This dialog box is used as an example in the exercise that follows.

As initially implemented, the opening dialog box is not resizable. This is indicated by its thick border, as seen below:



To make things more interesting, the opening dialog box will be given the capability to resize its contents based on its own size. This means, that as the dialog box grows or shrinks, any text and pictures within it grow or shrink proportionately.

These are the assumptions that were made prior to creating the dialog box:

Entity	Assumption
Window	The window is divided into two equal parts. The left side is used to display the bitmap. The right side is used to display the text and an OK push button.
Text	The text is to be centered on the right side of the window.
Text width and height	The right side of the window can hold a maximum number of lines (height) and a maximum number of characters (width) defined by the constants, LINES_DOWN and CHARS_ACROSS, respectively.
OK button	The OK push button is to be centered at the bottom of the right side of the window. Its size remains constant and is used to determine the minimum height and width of the window, since this button should always remain visible.

Exercise

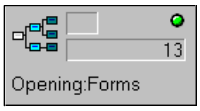
In the following exercise, you have the opportunity to examine the source code in the South Seas Adventures application responsible for creating and displaying the opening dialog box. During this exercise, you will:

- Modify the _OpeningDialog window entity to allow the opening dialog box to be resized
- Examine the source code necessary for resizing
- Examine the source code for displaying and dynamically resizing a bitmap
- Examine the source code for displaying and dynamically resizing text
- Examine the source code for dynamically positioning the OK push button
- Run the application to see the results of the generated source code

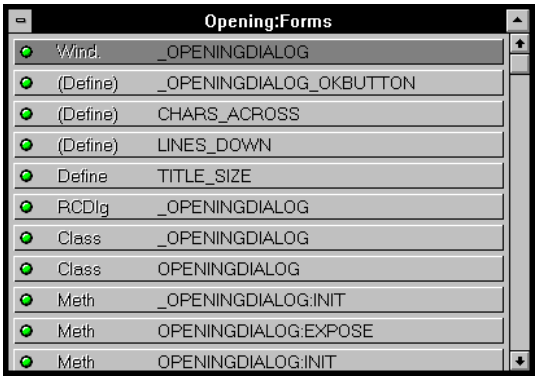
Making the Dialog Box Resizable

Initially, the opening dialog box was set up to be a fixed size; therefore, the first thing you will do in this lesson is make it resizable. The ability to resize a dialog box is controlled by one of its style properties. To change it, open the dialog box in the Window Editor as follows:

1. From the South Seas Adventures Module Browser, double-click on the Opening:Forms module button:

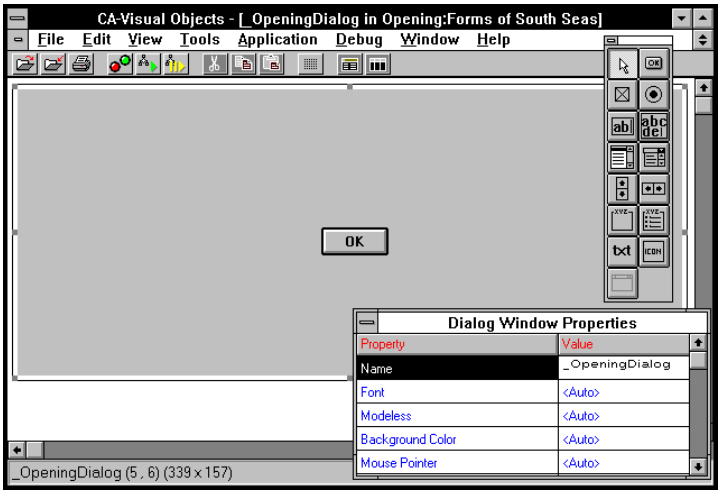


An Entity Browser is displayed:



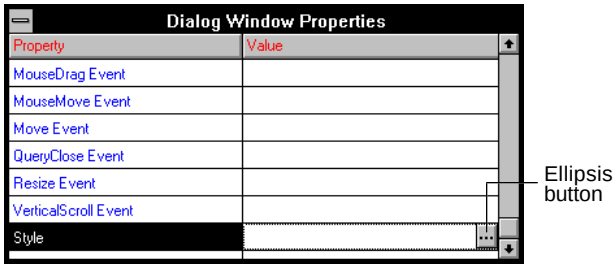
2. Double-click on the _OpeningDialog window entity.

3. The Window Editor, which was used to create the window and the OK button, is displayed:



4. Scroll through the Dialog Window Properties window and select the Style property.

5. Click on the Ellipsis button that appears in the value cell to the right of the Style property:



The Dialog Styles dialog box appears:

6. Click on the Resize radio button and choose OK.

   This enables you to resize the South Seas Adventures application opening dialog box.

7. Select the Save toolbar button.

8. Exit the Window Editor by double-clicking on its system menu.

The Resize Event

When you make a window resizable, as described in the previous steps, certain aspects of the physical resizing event are handled automatically. When the user attempts to resize the window, the system generates a Resize event that is, by default, handled by the Window:Resize() event handler method. However, if you have any special processing unique to your window, you must code a Resize() method of your own to handle it. (This has already been done for the OpeningDialog class.)

Note: The OpeningDialog class is a subclass created directly from the _OpeningDialog class, which was generated by the Window Editor. When working with code generated by CA-Visual Objects, it is wise to use this strategy of subclassing for making code changes. If you make changes directly to the generated code, you risk losing the changes each time the editor needs to regenerate code.

To see how the Resize event is handled for the opening dialog box, find the OpeningDialog:Resize() method in the Entity Browser and double-click on it.

The following code is loaded in the Source Code Editor:

```
METHOD Resize(oRSE) CLASS OpeningDialog

LOCAL oDim AS DIMENSION
LOCAL iMinHeight, iMinWidth AS INT

SUPER:Resize(oRSE)

// Put your changes here
```

```
// Screen size is based on size of the button
iMinHeight := SELF:oCCOkButton:Size:Height * 5
iMinWidth := SELF:oCCOkButton:Size:Width * 3

IF SELF:Size:Height < iMinHeight .OR.    ;
   SELF:Size:Width < iMinWidth
   // Don't let screen get too small
   SELF:Size := Dimension{MAX(iMinWidth, ;
      SELF:Size:Width), Max(iMinHeight,  ;
      SELF:Size:Height)}

ENDIF
// Repaint on every size since entire
// screen is proportional
SELF:Repaint()

RETURN NIL
```

The code checks the height and width of the window and, if necessary, resets it. To make sure that the OK push button is always visible in the dialog box, the size of the button was taken into consideration in the calculation:

```
iMinHeight := SELF:oCCOkButton:Size:Height * 5
iMinWidth := SELF:oCCOkButton:Size:Width * 3
```

From this calculation, you can see that the minimum height of the window is five times the height of the button. Similarly, the width of the window is set to three times the width of the button.

The following line forces the window to repaint each time the window is resized:

```
SELF:Repaint()
```

This forces an Expose event to occur. The Expose() event handler can then repaint our objects in the window, which is discussed in more detail later in this lesson.

Now that you have explored the Resize() method, it is time to move on to the code in which the bitmap is displayed in this dialog box. Before moving on, close the Source Code Editor by double-clicking on its system menu.

Using Bitmaps

The opening dialog box for the South Seas Adventures application displays a bitmap image from a .BMP file.

To display a bitmap from a .BMP file, you must:

- Declare the file as a resource
- Create a BitmapObject (a subclass of DrawObject)
- Create an Init() method for the Bitmap

Declaring a .BMP File as a Resource

A resource declaration statement is needed for accelerators, bitmaps, cursors, dialogs, icons, and menus. The code for these is usually generated by the associated visual editor. However, on occasion you must directly enter the code for the resource declaration via a RESOURCE statement in the Source Code Editor. Such is the case with bitmap images.

Note: When the application is built, all RESOURCE statements are sent directly to the Windows resource compiler.

To view the source code responsible for declaring the bitmap used in the opening dialog box:

1. Close the Opening:Forms Entity Browser, and select the App:Resources module by clicking on its module button:

2. Choose the Edit All Source toolbar button.

3. Find the RESOURCE SSABitmap entity, which reads as follows (assuming you have installed CA-Visual Objects to C:\CAVO):

```
RESOURCE SSABitmap Bitmap;
   c:\cavo\samples\ssatutor\files\ssa.bmp
```

4. Exit from the Source Code Editor by double-clicking on its system menu.

Tip: Use either the Find toolbar button or the Go to Entity toolbar button in the Source Code Editor to locate an entity quickly.

Creating a Bitmap Object

The next step is to create an object of the Bitmap class that refers directly to the resource declared in the previous steps. This has already been done in the South Seas Adventures application, but it is helpful to look at the source code to understand the connections.

Declaring a Bitmap Subclass: SSABitmap

At this point, the App:Resources Entity Browser should still be open.

1. Locate the SSABitmap class entity, and double-click on it to load it into the Source Code Editor.

   You see the following line of code:

   ```
   CLASS SSABitmap INHERIT Bitmap
   ```

2. Return to the Entity Browser by clicking on the Open toolbar button.
3. Locate the SSABitmap:Init() method and double-click on it.
4. Scroll through the Source Code Editor window to view the source code for both entities, which should look as follows:

   ```
   CLASS SSABitmap INHERIT Bitmap

   METHOD Init() CLASS SSABitmap
      SUPER:Init(ResourceID{"SSABITMAP"})

      RETURN SELF
   ```

This code is fairly straightforward. First, we have created a subclass of the Bitmap class. Then, we have defined an Init() method to be executed when objects of this subclass are instantiated.

Within the Init() method, note the use of the ResourceID class, which provides a unique identifier for a resource based on its name. You could declare your resources using unique identifiers rather than names, but working with resource names and converting them using the ResourceID class is much easier.

5. When you are finished looking at this source code, close the Source Code Editor by double-clicking on its system menu.

Instantiating an SSABitmap Object

After defining the class and instantiation code for the new bitmap, it is necessary to give the opening dialog box access to the bitmap. The most logical place to do this is from within the OpeningDialog:Init() method.

1. Close the App:Resources Entity Browser, and open the Entity Browser for the Opening:Forms module by double-clicking on its module button.
2. Find the OpeningDialog class entity, and double-click on it to view it in the Source Code Editor.

   You see the following code:

```
CLASS OpeningDialog INHERIT _OpeningDialog
   PROTECT LogoBitmap
```

3. Return to the Entity Browser by clicking on the Open toolbar button.
4. Find the OpeningDialog:Init() method and double-click on it.
5. Scroll through the Source Code Editor window to view the source code for both entities—it should look as follows:

```
CLASS OpeningDialog INHERIT _OpeningDialog
   PROTECT LogoBitmap

METHOD Init(oParent) CLASS OpeningDialog
   SUPER:Init(oParent)
   LogoBitmap := SSABitmap{}

   RETURN SELF
```

In the OpeningDialog:Init() method, the *LogoBitmap* instance variable (declared in CLASS OpeningDialog) is set to an SSABitmap object. Creating the bitmap in this manner allows you to create the object once, as part of the window instantiation, and then reuse it each time the window is redrawn.

6. When you are finished looking at this source code, close the Source Code Editor by double-clicking on its system menu.

Using a Bitmap Object to Draw a Bitmap on a Window

Now, let's look at the code in which the bitmap is actually drawn on the dialog box. Earlier, when we discussed resizing a window via the Resize event, we mentioned the Expose event that was triggered as a result of calling the Window:Repaint() method. The Expose event can now be viewed in greater detail.

The Expose Event

An Expose event occurs whenever the windows needs repainting. This can occur under any of the following circumstances:

- The window is first shown
- The window is partially uncovered by another window
- The window changes in size
- The window is being restored after being minimized
- A call is made to the window's Repaint() or RepaintBoundingBox() method

At this point, the Opening:Forms Entity Browser should still be open. Find the OpeningDialog:Expose() method, and double-click on it to load it into the Source Code Editor.

The following lines of code define the available size (adjusted to remove four pixels for each border) where the available width for the bitmap is the integer iMidWidth:

```
// Get screen height and width minus borders
iHeight:= SELF:Size:Height-8
iWidth:=SELF:Size:Width-8
iMidWidth:=INT(iWidth/2)
```

Now let's examine the SELF:Draw() line of code (located around line 14) which is responsible for drawing the SSABitmap:

```
// Draw bitmap sized and positioned
// relative to screen
SELF:Draw(BitmapObject{Point{2, 2},           ;
       Dimension{iMidWidth - 4, iHeight - 4}, ;
       LogoBitmap})
```

This is a fairly complicated line of code. Let's examine its individual components to get a better understanding of what is going on:

BitmapObject{...} Creates a bitmap draw object.

Point{2, 2} Defines the point, in pixels, at which to start drawing the bitmap object.

Dimension{iMidWidth-4, iHeight-4} Defines the pixel width and height of the object to be drawn. In this case, the width is half the window width less four pixels, and the height is the window height less four pixels. This ensures that the size of the bitmap is always relative to the size of the window.

LogoBitmap The bitmap object that was assigned when the opening dialog box was created.

SELF:Draw(...) Draws the logo on the dialog box.

Important! *You should always use the Expose() method when displaying DrawObject class entities. Also, use the Window:Draw() method to display your individual DrawObject. Never use the DrawObject:Draw() method directly, as it is called by the Window:Draw() method.*

Using Text Objects

The Expose() event handler method also has code for displaying the text on the dialog box. This is accomplished using the TextObject class, which is like BitmapObject in that it is a subclass of DrawObject.

You should still have the source code on your window at this point. Just below the line of code discussed previously for drawing the bitmap, you should see the following lines of code:

```
iFontWidth := INT(iMidWidth / CHARS_ACROSS)
iFontHeight := INT(iHeight / LINES_DOWN)
```

These lines calculate the width and height of the font, in canvas coordinates, used to display the text. The width is calculated as the window width divided by the number of characters allowed (CHARS_ACROSS). The height is calculated as the window height divided by the number of lines allowable (LINES_DOWN). In this way, the font is scaled based on the size of the window.

Note: Both CHARS_ACROSS (set to 30) and LINES_DOWN (set to 7) are defined as constants in this same module. If you wish, you can view them in the Source Code Editor by clicking on the Open toolbar button and then double-clicking the corresponding entity in the browser.

Next, the font is instantiated using the height and width calculated previously:

```
oTextFont := Font{FONTROMAN,     ;
   Dimension{iFontWidth, iFontHeight}}
```

Finally, the TextObject object is created and drawn on the dialog box. The code for doing this is similar to the code for drawing the bitmap that you saw earlier. The TextObject object is instantiated within the call to SELF:Draw(), and the starting point is determined dynamically, based on the font and window size:

```
sLine1 := "South Seas Adventures"
SELF:Draw(TextObject{Point{iMidWidth +  ;
   Int((CHARS_ACROSS-Len(sLine1))/2) *  ;
      iFontWidth, ;
   Int(Float(iHeight ) * .5)}, sLine1,  ;
      oTextFont})
```

Dynamic Positioning of Controls

The OK push button is the only entity on the window that is not scaled—although it could be. Since this is the only push button control on the window, it is best to fix its size as you could not afford to lose it to resizing.

This button, however, is dynamically positioned so that it is always visible. The positioning of the OK button is also handled within the Expose() event handler, which should still be on your window. Look immediately below the line of code discussed previously for drawing the text, and you see the following lines of code:

```
// Position push button
oCCOkButton:Origin := Point{     ;
   Int((Float(iWidth) * .75) -   ;
   (oCCOkButton:Size:Width / 2)), 10}
```

In this code, *oCCOkButton* is the name of the push button object. By calculating its *Origin* property based on the current width of the dialog box, the position of the OK push button is computed at runtime each time the dialog box is redisplayed.

Viewing the Results in the Application

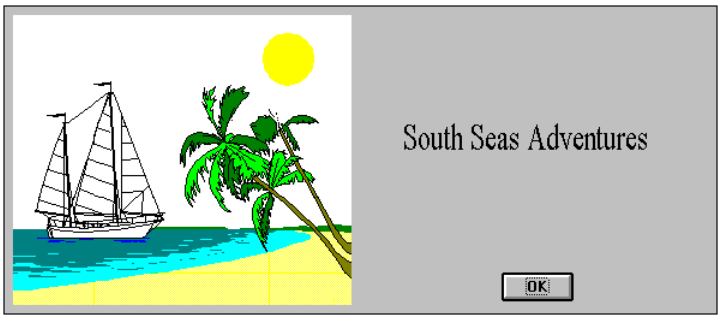
This concludes the overview of the source code used to control the opening dialog box. If you like, you can verify and examine the results in the South Seas Adventures application.

1. Close the Source Code Editor by double-clicking on its system menu.
2. Close the Opening:Forms Entity Browser by double-clicking on its system menu.

3. Even though you have not made any source code changes, you need to rebuild the application, using the Build toolbar button, because of the minor change you made earlier in the Window Editor.

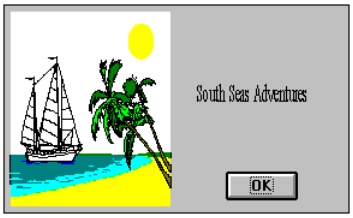
4. Run the South Seas Adventures application by clicking the Execute toolbar button.

   The opening dialog box is displayed:



   Note that it has a thin border, to indicate that it is now resizable.

5. Resize the dialog box by dragging the right border to the left. Note that everything except the push button is scaled down, as shown below:



6. Continue to resize the dialog box to see the results of the source code you have been reviewing. When you are done verifying the results, choose OK.
7. At the Login dialog box, choose the Cancel button.

Summary

In this lesson, you learned how to use the BitmapObject and TextObject classes, and how to scale objects created by these classes according to the window in which they reside. You also learned about the Expose and Resize events.

In the following lesson, you will learn how to use CA-RET to define, customize, preview, and print a report.