CHAPTER 8

CANVASES AND WINDOWS

Chapter Objectives

In this Chapter, you will learn about:

So far, your forms have been limited to the windows and canvases that Forms has created for you. As your forms get more complex, you will want to add additional windows and canvases and display them in response to interface or internal processing events.

This Chapter will cover the fundamentals of canvas and window objects, as well as a variety of methods for displaying them.

LAB 8.1

CANVAS AND WINDOW CONCEPTS

LAB OBJECTIVES

After this Lab, you will be able to:

- Understand Windows
- Understand Canvases

For a canvas and its items to be visible, it must be assigned to a window. By the same token, for a window to be visible, it must contain at least one canvas. Therefore, canvases and windows rely on each other to make themselves and other Forms objects appear to the user. The forms you have created so far have had one window and one canvas. The window (`WINDOW1`) was created by default when the new form was created. The canvas was created by the Layout Wizard and automatically assigned to `WINDOW1`. A single form can contain multiple instances of both canvases and windows, which you can create and define yourself.

WINDOWS

Like any other physical object in Forms, windows have properties that determine their size and position. Under the Functional property category, they have a series of window-specific properties that determine whether or not the window can be minimized, resized, closed, and so on.

There are two styles of windows in Forms: document and dialog. Document windows are commonly used for data entry and query screens; they are the main windows in an application. Dialog windows are used to deliver messages, display lists, or present the user with a specific task. On Microsoft Windows operating systems, there is a third style of window

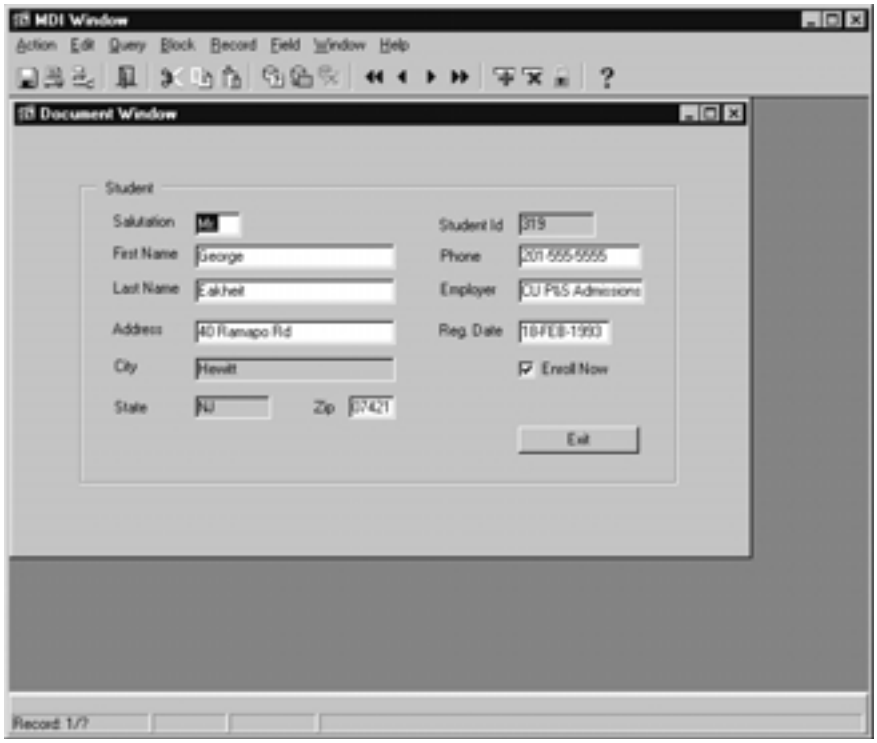

Figure 8.1 ■ An MDI window with a single document window.

called the Multiple Document Interface (MDI) window. The MDI window serves as a parent window to all of the other windows in a form. Figures 8.1 and 8.2 show examples of all three window styles. Note that the window titles in the figures indicate the window style.

Document and dialog windows are contained differently by the MDI window. Document windows are completely contained by their parent MDI window as illustrated in Figure 8.1. If a user were to drag the document window as far to the left as possible, it would disappear under the boundary of the MDI window. This is not the case with the dialog window (LOV) in Figure 8.2. It is not restricted by the MDI window's boundaries and can be moved completely outside of it.

Windows also have a modal property that helps determine how and when the user can leave the window. Navigation cannot leave a modal window until the user has completed whatever task the window calls for.

Figure 8.2 ■ A dialog window outside the boundaries of the MDI window.

■ *FOR EXAMPLE:*

In Figure 8.2, the LOV titled `Dialog Window` is modal. The user must either select from the LOV or click the `Cancel` button before going back to the window titled `Document Window`. The user cannot navigate to and from other windows. Modal windows give you the opportunity to present the user with a message or task and force them to respond to it before moving on.

If a window is modeless, the user can navigate to another window and leave the current window open and accessible. Modeless windows let you give your users the convenience of being able to switch from window to window.

Document windows are usually modeless (`Modal` property set to `No`) and dialog windows are usually modal (`Modal` property set to `Yes`). The `WINDOW1` object created by the Form Builder is a modeless document window.

DISPLAYING WINDOWS

There are a variety of methods for displaying and hiding windows at runtime. You can explicitly open windows by using window built-ins, you can set their properties, or you can simply navigate to them.

SHOW_WINDOW is a window-specific built-in that can be used to open all types of windows, be they modal or modeless, document or dialog.

■ FOR EXAMPLE:

You could open a window named MAINWIN with the following statement:

```
SHOW_WINDOW('MAINWIN');
```

Depending on the modality of MAINWIN and how its properties are set, the SHOW_WINDOW statement can produce different results in how the window is opened. It may display the window's content canvas or not. It may try to navigate to an item in the window or not. In the Exercises, you will experiment with some of the different outcomes of SHOW_WINDOW.

You can also open a window simply by navigating to an item in that window, or to a block that contains an item in the window. Keep in mind that for an item to "be in a window" it must be assigned to a canvas that is assigned to that window. If navigation has moved to a certain item and that item is not displayed or is not visible, Forms will make it visible. It will, among other things, display the canvas and window that item is assigned to. So by using built-ins to navigate to an item, you can open/display canvases and windows.

■ FOR EXAMPLE:

Assume you have the following code in a trigger:

```
GO_ITEM('BLOCK1.ITEM1');
```

Also assume that ITEM1 is on CANVAS1, which is assigned to MAINWIN. When Forms navigates to ITEM1, it must display it. So, if ITEM1 is not visible when the GO_ITEM built-in is executed, Forms will automatically show the canvas and window to make ITEM1 visible to the user.

There are numerous combinations of property settings and programmatic or navigational methods that affect the opening and closing of windows

and their canvases. In the Exercises, you will experiment with some of these combinations.

CANVASES

So far, you have worked with content canvases to display and position the items in your form. Content canvases are the most common canvas type because every window must have one as its main canvas. In the Exercises in the rest of this Chapter, you will experiment with two other types of canvases: stacked and toolbar. Also, in the "Test Your Thinking" section, you will attempt to build a form with a tabbed canvas. In this Lab, you will focus on some concepts that are common to all types of canvases.

CANVAS VIEWPORTS (VIEWS)

The terms "canvas view" and "viewport" are synonymous; viewport will be used in this interactive workbook. The viewport refers to the area of the canvas that is visible to the user. What this means is that the canvas object itself is not always entirely visible; only the area defined by its viewport is visible. Returning briefly to the painter's canvas analogy, think of a painting with a wide wooden frame. The frame obscures the

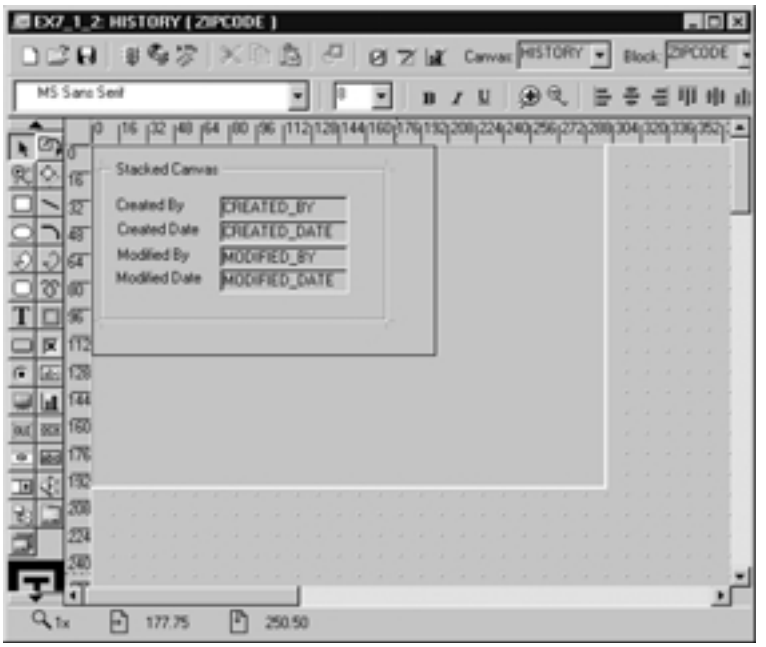

Figure 8.3 ■ A canvas and its viewport in the Layout Editor.

edges of the canvas, but the center of the canvas, the part with the painting, is visible. Think of the part that is visible as the viewport.

The size of the canvas and the size of its viewport are determined by properties. Figure 8.3 shows what a canvas and its viewport look like in the Layout Editor.

The viewport in Figure 8.3 is represented by the thin black line that surrounds the items and the frame. The canvas is represented by the larger rectangular area. The canvas ends where the surface of the Layout Editor appears to have raised dots. On a painter's canvas, the painting is rarely ever behind the frame, or in any way hidden from the viewer. This is not necessarily the case with Forms canvases. It is common to place some items outside the viewport so that they are not initially visible. Then, as the user scrolls or navigates the form, the viewport can move to expose different parts of the canvas. In the "Test Your Thinking" section of this Chapter, you will be asked to create a canvas whose viewport will move to show and hide items and create a scrolling effect.

DISPLAYING CANVASES

The rules for displaying and hiding canvases are similar to those for windows. A canvas can be displayed either by using built-ins to open it explicitly or by navigating to an item on the canvas. If you choose to display a canvas by navigating to it, you could use the same `GO_ITEM` built-in explained in the example above. If you choose to display a canvas explicitly, you could use the `SHOW_VIEW` built-in and pass it the canvas' name. The code would look like this for a canvas named `CANVAS1`:

```
SHOW_VIEW('CANVAS1');
```

LAB 8.1 EXERCISES

8.1.1 UNDERSTAND WINDOWS

Open the form `EX08_01.fmb` in the Form Builder. Run the form and issue a query.

a) Can you drag the window titled `Main Window` out of the MDI window? What does this tell you about its window style?

Click the Alert button.

b) Can you drag the alert out of the MDI window? What does this tell you about its window style?

c) Can you navigate to the form without closing the alert? What does this tell you about its modality?

Click the OK button to close the alert. Click the Window button.

d) What style is this window? Is it modal?

e) Have any properties been set for the MDI window? How can you tell?

Exit the running form and return to the Form Builder. Open the WHEN-BUTTON-PRESSED trigger for the CONTROL.SHOW_HIST button.

f) Is this built-in opening the window explicitly?

Comment out the current code in the SHOW_HIST button's WHEN-BUTTON-PRESSED trigger and then add the code shown below:

```
--GO_ITEM('CONTROL.HIDEWIN);
SHOW_WINDOW('SECONDWIN');
```

Run the form and click the Window button.

g) Did the window open? Are there any items visible? What does this mean about opening a window with navigation vs. doing it programmatically?

Reenter the SHOW_HIST button's WHEN-BUTTON-PRESSED trigger and change it back so it appears as it does below:

```
GO_ITEM('CONTROL.HIDEWIN);
--SHOW_WINDOW('SECONDWIN');
```

h) Which built-ins are being used in the HIDE_HIST button's WHEN-BUTTON-PRESSED trigger?

Comment out HIDE_WINDOW('SECONDWIN'); in the HIDE_HIST button's WHEN-BUTTON-PRESSED trigger. Run the form and press the Window button. Now click the Close button.

i) Did navigation return to the ZIP item on the MAINWIN? Did the SECONDWIN close? What does this tell you about navigation in and out of modeless windows?

8.1.2 Understand Canvases

Open the EX08_02.fmb form in the Form Builder.

a) How many canvases are there and what are their names and types?

b) Which window is the `ZIPCODE` canvas assigned to? Which window is the `HISTORY` canvas assigned to?

Open the `STACKED` button's `WHEN-BUTTON-PRESSED` trigger.

c) How big is the `HISTORY` canvas? How big is its viewport?

d) How is the `HISTORY` canvas being displayed?

Run the form and test the `Stacked Canvas` button to see how the stacked canvas will behave.

LAB 8.1 EXERCISE ANSWERS

8.1.1 ANSWERS

Open the form `EX08_01.fmb` in the Form Builder. Run the form and issue a query.

a) Can you drag the window titled `Main Window` out of the MDI window? What does this tell you about its window style?

Answer: No you cannot. It is a document window.

Document windows are the most common windows in typical Forms applications. Most data entry and transaction-oriented work will be done on canvases that are displayed within document windows. In this form, and in most of the forms you will create using this book, you will only have one document window per module. However, it is common to build forms that have multiple document windows. These windows can be configured to be open simultaneously so that users can toggle back and

forth between them, just like in MSWord and other typical windowing applications.

Click the `Alert` button.

b) Can you drag the alert out of the MDI window? What does this tell you about its window style?

Answer: Yes you can. It is a dialog window.

Alerts are good examples of simple dialog windows since they allow the user to read the alert message and then drag it completely out of the way to view the form before responding.

c) Can you navigate to the form without closing the alert? What does this tell you about its modality?

Answer: No you cannot. It is a modal window.

Dialog-style windows are often implemented as modal. They are windows that require some sort of response from the user before regular processing can continue. The alert in `EX08_01.fmb` will not let the user continue until they have clicked the `OK` button. In this case, the alert was merely informational. The response required can be seen as an "OK, I read the alert." But other alerts and modal dialogs require that the user make a decision or complete some tasks.

■ *FOR EXAMPLE:*

Figure 8.4 shows the `LOV Column Mapping` dialog from the properties of an LOV. This modal dialog requires that the user enter some information before continuing.

Figure 8.4 ■ The `Lov Column Mapping` dialog.

go to contents

Click the OK button to close the alert. Click the Window button.

d) What style is this window? Is it modal?

Answer: It is a dialog window. No, it is modeless.

Although many dialogs are modal, it is not required that they be that way. In this case, the window titled Second Window is similar to a help window. No input is required from the user; it is merely displaying the history of the current record.

e) Have any properties been set for the MDI window? How can you tell?

Answer: Yes, the Title *property has been set.*

The MDI window is not visible in the Object Navigator, nor are its properties visible in the Property Palette. However, it does have properties and they can be adjusted programmatically using the SET_WINDOW_PROPERTY built-in. In this case, the PRE-FORM trigger sets three properties for the MDI window: Height, Width, and Title. Keep in mind that the MDI window is only available on Windows platforms.

Open the PRE-FORM trigger and look at the code. Note that the built-in refers to the MDI window a bit differently than it does other windows.

FOR EXAMPLE:

The statements in the PRE-FORM trigger look like this:

```
SET_WINDOW_PROPERTY(FORMS_MDI_WINDOW, WIDTH, 550);
```

FORMS_MDI_WINDOW is a Forms constant. It is not possible to refer to the MDI window by name in single quotes or with an object ID.

Exit the running form and return to the Form Builder. Open the WHEN-BUTTON-PRESSED trigger for the CONTROL.SHOW_HIST button.

f) Is this built-in opening the window explicitly?

Answer: No, the window is being opened by navigation.

This is a fairly straightforward method for opening a window and its canvas. Once you have told Forms which item it should navigate to, it does the rest of the work by opening the window and the appropriate canvas.

Comment out the current code in the SHOW_HIST button's WHEN-BUTTON-PRESSED trigger and then add the code shown below:

```
--GO_ITEM('CONTROL.CLOSE_HIST);
SHOW_WINDOW('SECONDWIN');
```

Run the form and click the Window button.

g) Did the window open? Are there any items visible? What does this mean about opening a window with navigation vs. doing it programmatically?

Answer: Yes, the window opened. No, there are no items visible.

In Question f, navigation automatically opened the window and canvas. Here you can see that opening the window does not automatically display the canvas or its items. To get the canvas and its items to appear, you would have had to add another line of code to the trigger or adjust one of the window's properties.

■ *FOR EXAMPLE:*

To get the canvas and its items to open along with the window, you could change the code to the following:

```
SHOW_WINDOW('SECONDWIN');
SHOW_VIEW('SECONDCAN');
```

In this case, you are explicitly opening both the window and its canvas. The other option is to use properties to manipulate the window and canvas. In that case, you would set the SECONDWIN window's Primary Canvas property to SECONDCAN. By doing this, SECONDCAN will open automatically when SECONDWIN is opened.

Reenter the SHOW_HIST button's WHEN-BUTTON-PRESSED trigger and change it back so it appears as it does below:

```
GO_ITEM('CONTROL.HIDE_HIST);
--SHOW_WINDOW('SECONDWIN');
```

h) Which built-ins are being used in the HIDE_HIST button's WHEN-BUTTON-PRESSED trigger?

Answer: The GO_ITEM *built-in, which is navigational, and the* HIDE_WINDOW *built-in, which explicitly closes the window.*

i) Did navigation return to the ZIP item on the MAINWIN? Did the SECONDWIN close? What does this tell you about navigation in and out of modeless windows?

Answer: Yes, navigation returned to the ZIP *item. No, the* SECONDWIN *did not close.*

SECONDWIN is a modeless window. Therefore, simply navigating out of the window did not close it. Why is this? Modeless windows allow users to work on two or more tasks simultaneously. A user can be adding a student enrollment in one window and adjusting another student's address in another. The user would want to be able to go back and forth from window to window without having either close.

If SECONDWIN had been a modal window, then the behavior would have been different. The GO_ITEM built-in would have closed SECONDWIN without needing the HIDE_WINDOW built-in. Why is this? By definition, a modal window cannot be navigated out of or closed until its task has been completed or perhaps canceled. If the trigger is forcing navigation with the GO_ITEM built-in, then Forms will assume that the modal window's job is done and will close it.

The methods for opening and closing windows may seem a bit confusing. A good rule of thumb is to try to navigate to a window or a canvas to open it. If the window does not have any navigable items, or if you wish to simply open the window without navigating to it, use the SHOW_ built-ins.

8.1.2 ANSWERS

Open the EX08_02.fmb form in the Form Builder.

a) How many canvases are there and what are their names and types?

Answer: There are two canvases: ZIPCODE *and* HISTORY. ZIPCODE *is a content canvas and* HISTORY *is a stacked canvas.*

b) Which window is the ZIPCODE canvas assigned to? Which window is the HISTORY canvas assigned to?

Answer: Both canvases are assigned to MAINWIN.

A canvas must be assigned to a window to be visible. A stacked canvas must always be stacked on a content canvas to be visible. Therefore, both canvases are assigned to the same window. It is technically possible to display a stacked canvas in a window without a content canvas, but it is not recommended.

Canvases, be they stacked or content, do not have to be explicitly assigned to windows at design-time. Instead, they can be assigned programmatically at run-time.

c) How big is the HISTORY canvas? How big is its viewport?

Answer: The Height *and* Width *properties of the* HISTORY *canvas are* 200, 300. *Its view, or viewport, is* 122,200.

The properties that govern the size of the actual canvas are under the Physical category and are called Height and Width. The properties that govern the size of the viewport are under Viewport category and are called Viewport Height and Viewport Width.

In this case, as in many, the canvas is far larger than the viewport. It is quite common to place items on the canvas but not include them in the viewport.

Open the STACKED button's WHEN-BUTTON-PRESSED trigger.

d) How is the HISTORY canvas being displayed?

Answer: It is being displayed programmatically using the SET_VIEW_PROPERTY *built-in.*

The same rules that apply to windows apply to canvases. It is easiest to display a canvas using navigation, but it is possible and sometimes necessary to display a canvas programmatically. In this case, it was necessary to display the HISTORY canvas programmatically because it does not contain any navigable items. As an alternate and equally effective method to using the SET_ built-in, the SHOW_VIEW and HIDE_VIEW built-ins would have worked equally well here also.

LAB 8.1 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions:

1) Which of the following is not a window style?
- **a)** ____ Dialog
- **b)** ____ Document
- **c)** ____ Content
- **d)** ____ MDI

2) Which of the following is true about document windows?
- **a)** ____ They have at least two content canvases
- **b)** ____ They can be dragged outside the MDI window
- **c)** ____ They are well-suited for data entry and query screens
- **d)** ____ They cannot be opened at run-time

3) Which of the following is false?
- **a)** ____ Alerts are dialog windows
- **b)** ____ LOVs are dialog windows
- **c)** ____ Dialog windows have at least one content canvas
- **d)** ____ Dialog windows cannot be modal

4) What does the `SHOW_WINDOW` built-in do?
- **a)** ____ It manages the MDI window
- **b)** ____ It displays windows
- **c)** ____ It responds to Open Window events
- **d)** ____ a & b

5) What will happen if the `GO_ITEM` built-in is passed a non-navigable item?
- **a)** ____ Navigation will not complete and the form will return an error
- **b)** ____ Forms will navigate to it anyway
- **c)** ____ Forms will open its window but not navigate to it
- **d)** ____ All canvases will close

6) What are some of the methods for displaying a canvas?
- **a)** ____ Navigate to an item on the canvas
- **b)** ____ Use the `SHOW_VIEW` built-in
- **c)** ____ Set the `MDI Windows Visible` property to `Yes`
- **d)** ____ a & b

7) Which of the following is not true about the viewport of a content canvas?
- **a)** ____ It is the same size as its window
- **b)** ____ It cannot contain stacked canvases
- **c)** ____ Items outside the viewport will not be visible at run-time
- **d)** ____ It can be adjusted in the Layout Editor

Quiz answers appear in Appendix A, Section 8.1.

LAB 8.2

CONTENT CANVASES AND WINDOWS

LAB OBJECTIVES

After this Lab, you will be able to:

- Create a Content Canvas and Window

While content canvases have viewports, they do not have properties to set the size of their viewports. The size of a content canvas' viewport is the same as the size of the window that it occupies.

FOR EXAMPLE:

Assume that `CANVAS1` is the content canvas for `WINDOW1`. If `WINDOW1`'s `Height` and `Width` properties are set to `200`, `200`, then this will be the size of `CANVAS1`'s viewport. To set the size of a content canvas' viewport, you must either:

1) Use the `Height` and `Width` properties of the window it is assigned to.
2) Adjust it visually in the Layout Editor.

In the Exercise for this Lab, you are going to walk through the process of creating a content canvas and window. Your objective will be to duplicate the functionality of the form `EX08_01.fmb`.

First you are going to create a basic form using the wizards. Then you will create a second canvas and window. You will use these to display the values of the audit columns. You will then create a button to show this window and experiment with showing it and hiding it with navigation.

go to contents

LAB 8.2 EXERCISES

8.2.1 CREATE A CONTENT CANVAS AND WINDOW

Use the wizards to quickly create a form based on the `STUDENT` table. `Enforce data integrity` should be **unchecked**. Include the audit columns in the block, but do not display them on the canvas. Give the canvas a `Form`-style layout.

Rename the window to `MAINWIN`.

a) How can you create another content canvas? Should you use the Object Navigator or the Layout Editor?

Rename the canvas `HISTORY`.

b) Do you want `HISTORY` to be a content canvas for `MAINWIN`? Why not?

c) What steps should you take to create a window and assign `HISTORY` as its content canvas?

Rename the window `HISTWIN`. Users should be able to drag `HISTWIN` outside the MDI window, and they should not be able to access other windows while it is open.

d) Which of the `HISTWIN` window's properties should you set to make the above happen?

go to contents

e) Should you assign the audit items to the HISTORY canvas or to the HISTWIN window? What's the easiest way to do this?

f) Can you view both canvases at the same time? How?

g) What should the item type for the audit columns be since you don't want users to be able to navigate to them or change their values?

Arrange the audit items and their prompts on the HISTORY canvas so that the layout is similar to that in Figure 8.5.

Create a non-base table block and name it CONTROL. Create two buttons in the CONTROL block and name them SHOW_HIST and HIDE_HIST. Position SHOW_HIST on the STUDENT canvas in the lower right-hand corner. Position HIDE_HIST on the HISTORY canvas below the items in the center. Set the Key Board Navigable and Mouse Navigable properties for both buttons to No.

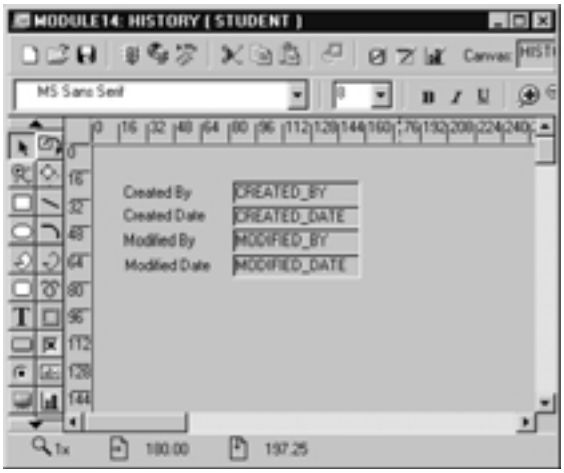

Figure 8.5 ■ The audit items arranged on the HISTORY canvas.

h) What navigation built-in could you use to open the HISTWIN window and its canvas?

i) Could you pass this built-in CREATED_BY as its parameter? Why not?

j) What is the only navigable item on the HISTORY canvas?

Create a WHEN-BUTTON-PRESSED trigger for the SHOW_HIST button with the following code:

```
GO_ITEM('CONTROL.HIDE_HIST');
```

Create another WHEN-BUTTON-PRESSED trigger for the HIDE_HIST button with the following code:

```
GO_ITEM('STUDENT.STUDENT_ID);
HIDE_WINDOW('HISTWIN');
```

Run the form and test the buttons.

k) Which object's X and Y Positions and Height and Width properties should you adjust to position and size the history information a bit better?

Try a Height of 130 and a Width of 190 to size the object. Adjust the X and Y Positions to taste, but try not to obscure any of the items on the STUDENT canvas.

go to contents

Run the form to test the size and positioning properties.

Navigate to STUDENT.ADDRESS and click the SHOW_HIST button. Now click the CLOSE_HIST button. Note that navigation did not return to STUDENT.ADDRESS, but to STUDENT.ZIP.

l) When the Button Pressed event occurs, you want to capture the value of the current item so that you can navigate back to it when the HISTWIN window closes. What system variable can you use to do this?

Change the code in the SHOW_HIST button to use a global variable. The code should now look like this:

```
:global.last_item := :SYSTEM.CURSOR_ITEM;
GO_ITEM('CONTROL.HIDE_HIST');
```

m) How can you use the global variable in the WHEN-BUTTON-PRESSED trigger for HIDE_HIST so that it will return to the item that called it?

n) Will these windows, canvases, buttons, and triggers be useful in other forms that use the audit columns? Will the triggers work as they are written now?

Save the form as R_CONCANV.fmb.

LAB 8.2 EXERCISE ANSWERS

8.2.1 ANSWERS

a) How can you create another content canvas? Should you use the Object Navigator or the Layout Editor?

Answer: You should use the Object Navigator.

There is no tool on the Layout Editor's Tool Palette to create content canvases, so you must use the Object Navigator. It is possible to use the Layout Wizard, but that is probably more trouble than it is worth. Note that when you use the Object Navigator, its default behavior is to create canvases with `Canvas Type` set to `Content`.

Rename the canvas `HISTORY`.

b) Do you want `HISTORY` to be a content canvas for `MAINWIN`? Why not?

Answer: No you do not.

In this Exercise, you want to duplicate the look and feel of the `EX08_01.fmb` form that you experimented with earlier. In that form, there were two content canvases, each assigned to their own windows.

It is possible to assign two content canvases to the same window, but only one can be visible at a time. You would use either navigation built-ins or explicit canvas built-ins to switch from one content canvas to the other.

c) What steps should you take to create a window and assign `HISTORY` as its content canvas?

Answer: Create a window and then set the `HISTORY` *canvas'* `Window` *property to the name of the window.*

d) Which of the `HISTWIN` window's properties should you set to make the above happen?

Answer: `Window Style` *should be set to* `Dialog` *and* `Modal` *should be set to* `Yes`.

e) Should you assign the audit items to the `HISTORY` canvas or to the `HISTWIN` window? What's the easiest way to do this?

Answer: You should assign them to the canvas. Select all of the audit items in the Object Navigator and set their `Canvas` *property to* `HISTORY`.

go to contents

Remember, items are assigned to canvases, and then canvases are assigned to windows.

f) Can you view both canvases at the same time? How?

Answer: Yes you can. You can open two instances of the Layout Editor.

You can have as many instances of the Layout Editor open as you'd like, which can make comparing canvases, especially stacked canvases, easier. You can also toggle from canvas to canvas using the `Canvas` list item on the Layout Editor's Utility toolbar.

In Lab 8.3, you will learn about another Layout Editor feature that allows you to view a content canvas with all of its stacked canvases stacked on top of it.

g) What should the item type for the audit columns be since you don't want users to be able to navigate to them or change their values?

Answer: The item type should be display item.

h) What navigation built-in could you use to open the `HISTWIN` window and its canvas?

Answer: You could use the `GO_ITEM` *built-in.*

You could also have used the `GO_BLOCK` built-in to navigate to the `CONTROL` block. `HIDE_HIST` is the first item in the block, so navigation would have succeeded.

i) Could you pass this built-in `CREATED_BY` as its parameter? Why not?

Answer: No, because it is a display item, it is not navigable.

The `GO_ITEM` built-in is only valid for items that are navigable. Display items are not navigable, so the navigation would have failed if you had tried to pass `CREATED_BY` into the `GO_ITEM` built-in.

j) What is the only navigable item on the `HISTORY` canvas?

Answer: The `HIDE_HIST` *button.*

Even though the `Keyboard Navigable` property was set to `No`, `SHOW_HIST` is considered a navigable item and the `GO_ITEM` built-in will succeed. Interestingly, as you learned in the previous question, this does not apply to display items since by definition they are non-navigable and have no `Keyboard Navigable` property.

go to contents

Again, it would be possible to use SHOW_WINDOW instead of GO_ITEM. But since the HIDE_HIST button is a navigable item, it is easier to simply navigate to it.

Run the form and test the buttons.

k) Which object's X and Y Positions and Height and Width properties should you adjust to position and size the history information a bit better?

Answer: The HISTWIN *window's properties.*

Since the HISTORY canvas is a content canvas, its sizing properties are ignored by the Form Builder and do not affect the size or position of the canvas at run-time. This is because the content canvas inhabits the entire surface of its window. Therefore, the size and position properties that you set for the window, in this case HISTWIN, are what dictate the size and positioning of the canvas.

l) When the Button Pressed event occurs, you want to capture the value of the current item so that you can navigate back to it when the HISTWIN window closes. What system variable can you use to do this?

Answer: :SYSTEM.CURSOR_ITEM.

m) How can you use the global variable in the WHEN-BUTTON-PRESSED trigger for HIDE_HIST so that it will return to the item that called it?

Answer: See discussion below.

The code should be:

```
GO_ITEM(:global.last_item);
HIDE_WINDOW('HISTWIN');
```

The global variable created and assigned in the SHOW_HIST trigger is referenced in the HIDE_HIST trigger so that the form will know which item to return to.

Global variables are user-defined variables that, like other variables, can hold values that need to be referenced later on. However, global variables are special in four ways:

1) They don't have to be declared in a DECLARE statement of a PL/SQL block.
2) They are assigned as they are created.

3) They can be referenced outside the PL/SQL block they were created in.
4) If multiple forms are open, they can all reference the global variable.

■ FOR EXAMPLE:

In this case, the global variable is not declared in a `DECLARE` section of a PL/SQL block; it is assigned a value as it is created. It is created in one trigger and referenced in another.

n) Will these windows, canvases, buttons, and triggers be useful in other forms that use the audit columns? Will the triggers work as they are written now?

Answer: Yes they will.

All of the objects that create the `Record History` window could be copied immediately to another form that has the audit columns in its data blocks. A little work would be required to assign and position the audit items on the `HISTORY` canvas, but once done, the objects would be completely usable. Why does this matter? Now that you have designed a way to let the user view the contents of the audit columns, you can apply it to every form you create that includes the audit columns. This way, you won't have to redo it for every form.

In Chapter 9, "Reusable Objects," you will learn a way to group these objects so that you can move them from form to form quickly and easily.

Save the form as `R_CONCANV.fmb`.

LAB 8.2 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions:

1) Two content canvases cannot appear in the same window.
- **a)** ____ True
- **b)** ____ False

2) How can a content canvas be created?
- **a)** ____ By using the Object Navigator
- **b)** ____ By using the Layout Wizard
- **c)** ____ By using the Layout Editor
- **d)** ____ a & b
- **e)** ____ a, b, & c

go to contents

3) Which of the following is true about a content canvas?

a) ____ It must be in a modal window
b) ____ It must be in a modeless dialog window
c) ____ It is contained by a window
d) ____ It can be visible without a window

4) Windows cannot be closed with navigation.

a) ____ True
b) ____ False

Quiz answers appear in Appendix A, Section 8.2.

LAB 8.3

STACKED CANVASES

LAB OBJECTIVES

After this Lab, you will be able to:

- Create and Display Stacked Canvases

Stacked canvases are never the sole canvas in a window. They are always stacked on top of other canvases, and partially or completely obscure those canvases when displayed at run-time. To stack a stacked canvas, you must position it relative to the content canvas that it is stacked upon. Figure 8.6 shows both a stacked and content canvas in the Layout Editor.

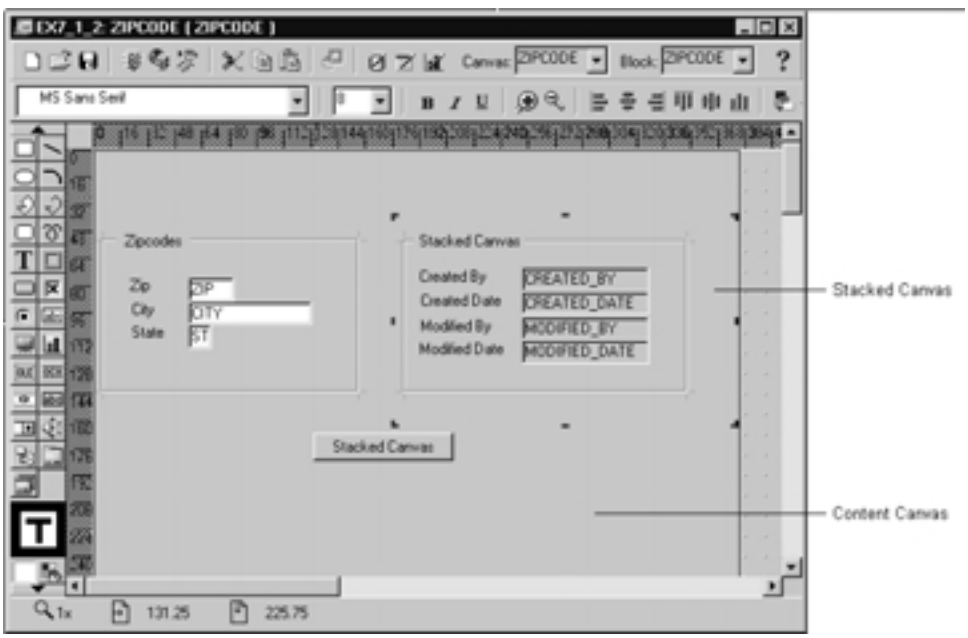

Figure 8.6 ■ A stacked canvas positioned on a content canvas in the Layout Editor.

go to contents

The stacked canvas' viewport is what is visible here. The actual stacked canvas object might be much larger. In this example, we see only one stacked canvas, but it is possible to have multiple stacked canvases in a single window, stacked upon a single content canvas. They could be stacked on different portions of the content canvas, or they could be stacked on top of one another.

Stacked canvases have `Height` and `Width` properties just like content canvases. They also have properties to determine the size and positioning of their viewports. The viewport will be positioned in two ways.

1) Relative to the stacked canvas itself.
2) Relative to the content canvas it is stacked upon.

The `Viewport X Position on Canvas` and `Viewport Y Position on Canvas` properties determine where the viewport will be positioned on the stacked canvas itself.

A stacked canvas has two other properties called `Viewport X Position` and `Viewport Y Position`. These position the canvas relative to the content canvas.

■ FOR EXAMPLE:

If the `Viewport X Position` of a stacked canvas is set to `40` (in points), and its `Viewport Y Position` is set to `178`, that means that its upper left-hand corner will be at those coordinates on the content canvas as illustrated in Figure 8.7.

In form `EX08_02.fmb`, you saw how a stacked canvas could be shown and hidden programmatically. In this example, there is only one stacked canvas, but there could have been many stacked on top of one another to simulate pages. A wizard can also be built using multiple stacked canvases.

■ FOR EXAMPLE:

Assume you are going to create a wizard object to add student records and then add enrollment records for those students. The wizard will have three or more wizard pages to step the user through adding a student's name, address, employment information, enrollment information, and so on. These pages will be implemented as stacked canvases and will be positioned on top of one another. The underlying content canvas will contain the wizard's `Back`, `Next`, and `Finish` buttons. You will use navigation or `SHOW_VIEW` and `HIDE_VIEW` built-ins to switch from page to page.

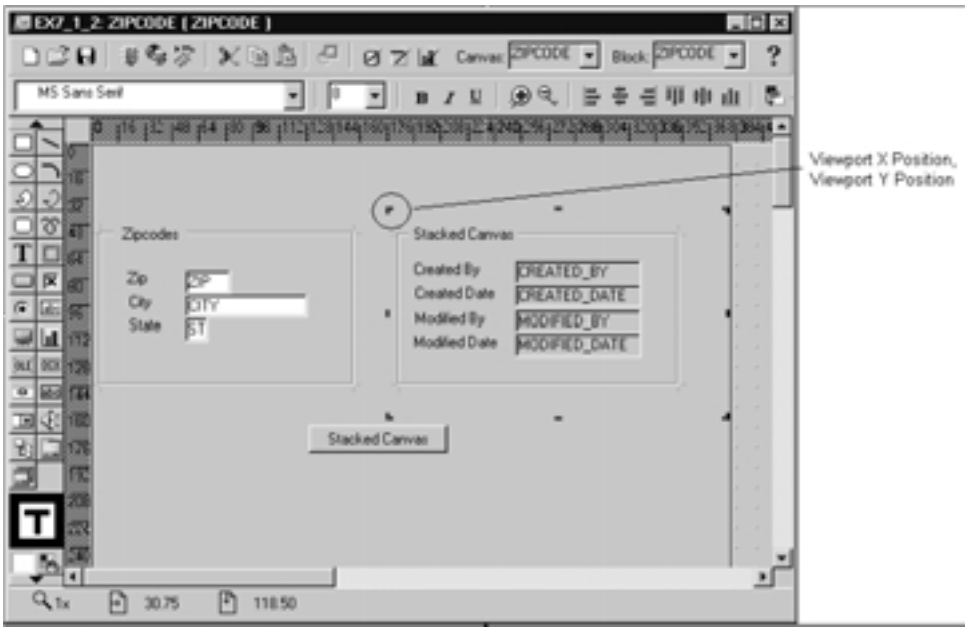

Figure 8.7 ■ Viewport `X` and `Y` `Positions` for the stacked canvas.

It is also common to use stacked canvases to simulate scrolling views. This can be useful if you have items that you'd like to display in tabular fashion but there are too many to comfortably fit on a normal-sized canvas. In the "Test Your Thinking" section, you will create a scrolling stacked canvas to accommodate all of the items in a `SECTION` block.

LAB 8.3 EXERCISES

8.3.1 CREATE AND DISPLAY STACKED CANVASES

In this Lab, you will create a stacked canvas called `INSTRUCTOR`. You will position `INSTRUCTOR` on a content canvas so that it covers some items. Then, you will use a button to programmatically show and hide the canvas.

Open form `EX08_03.fmb` in the Form Builder. Note that there are four blocks: `SECTION`, `ENROLLMENT`, `INSTRUCTOR`, and `CONTROL`. The `SECTION` and `ENROLLMENT` items are positioned on the `SECTION` content canvas. The `INSTRUCTOR` items have not been positioned yet.

a) What are three Form Builder tools you could use to create the stacked canvas?

The stacked canvas will ultimately lie on top of the content canvas.

b) With this in mind, which of the tools that you listed in Question a might be the best for creating and simultaneously positioning the stacked canvas?

Create the stacked canvas using the `Stacked Canvas` tool from the Layout Editor's Tool Palette. Make sure you completely cover the `ENROLLMENT` items on the content canvas, but do not obscure or cover the frame. Name the new stacked canvas `INSTRUCTOR`.

c) Are you able to see both the content canvas and stacked canvas in the Layout Editor? Is part of the content canvas obscured?

Select `View | Stacked Views` from the Main Menu. When the `Stacked/Tab Canvases` dialog opens, de-select `INSTRUCTOR` and click the `OK` button.

d) Are you still able to see both canvases? What does this tell you about the `View | Stacked Views` feature?

Select `View | Stacked Views` from the Main Menu again. When the `Stacked/Tab Canvases` dialog opens, re-select `INSTRUCTOR` and click the `OK` button so that `INSTRUCTOR` is visible on top of `SECTION`.

In the Object Navigator, click the icon next to INSTRUCTOR to open another Layout Editor window.

e) How can you position the items in the INSTRUCTOR block on the INSTRUCTOR canvas?

Run the form and issue a query. The Change View button will not work yet, so use Block | Next Block from the Main Menu to test the stacked canvas. Exit the form and return to the Form Builder.

f) Which properties can you adjust to make the stacked canvas and its items look better? For example, isn't the stacked canvas set a bit lower than the content canvas?

g) Which of the stacked canvas' properties can you change so that the canvas is not visible when the form opens?

Open the WHEN-BUTTON-PRESSED trigger for the CHANGEVIEW button.

h) What trigger code could you write to change the canvas' Visible property? Your trigger should first evaluate whether or not the canvas is visible. If the canvas is not visible, use a built-in to show it. If it is visible, use a built-in to hide it. Use conditional logic and GET_ and SET_ built-ins to accomplish this.

Run the form, issue a query, and test the Change View button.

i) What are two other built-ins you could use to hide and show the canvas? These should not be `GET_` or `SET_` built-ins, and they should not cause navigation.

Run the form, issue a query, and test the `Change View` button. Exit the form and return to the Form Builder.

Replace the built-ins you used for Question i with `GO_BLOCK('INSTRUCTOR')` and `GO_BLOCK('ENROLLMENT')` statements.

Run the form, issue a query, and test the `Change View` button.

j) How is the behavior of the form different now that you are using navigational built-ins instead of the built-ins you used in Questions h and i?

LAB 8.3 EXERCISE ANSWERS

8.3.1 ANSWERS

In this Lab, you will create a stacked canvas called `INSTRUCTOR`. You will position `INSTRUCTOR` on a content canvas so that it covers some items. Then, you will use a button to programmatically show and hide the canvas.

Open form `EX08_03.fmb` in the Form Builder. Note that there are four blocks: `SECTION`, `ENROLLMENT`, `INSTRUCTOR`, and `CONTROL`. The `SECTION` and `ENROLLMENT` items are positioned on the `SECTION` content canvas. The `INSTRUCTOR` items have not been positioned yet.

a) What are three Form Builder tools you could use to create the stacked canvas?

Answer: You could use the Object Navigator, Layout Editor, or Layout Wizard.

The stacked canvas will ultimately lie on top of the content canvas.

b) With this in mind, which of the tools that you listed in Question a might be the best for creating and simultaneously positioning the stacked canvas?

Answer: The Layout Editor.

go to contents

By using the Layout Editor, you will be able to treat the stacked canvas like an item. You will be able to position the cursor where you would like the canvas to begin and stretch out the canvas until it reaches the size you desire.

Create the stacked canvas using the `Stacked Canvas` tool from the Layout Editor's Tool Palette. Make sure you completely cover the `ENROLLMENT` items on the content canvas, but do not obscure or cover the frame.

c) Are you able to see both the content canvas and stacked canvas in the Layout Editor? Is part of the content canvas obscured?

Answer: Yes, you can see both and yes, part of the content canvas is obscured.

Select `View | Stacked Views` from the Main Menu. When the `Stacked/Tab Canvases` dialog opens, de-select `INSTRUCTOR` and click the `OK` button.

d) Are you still able to see both canvases? What does this tell you about the `View | Stacked Views` feature?

Answer: No, you cannot see both.

The `View | Stacked Views` feature lets you view the size and position of the stacked canvases on their content canvases. The stacked canvas and its objects are accessible in this mode so that you can select and adjust their positions.

In the previous Lab, you learned that you can have two instances of the Layout Editor open. This can also be helpful in that it lets you see the stacked canvas on its own. Additionally, the Stacked Views mode shows only the stacked canvas' viewport.

Select `View | Stacked Views` from the Main Menu again. When the `Stacked/Tab Canvases` dialog opens, re-select `INSTRUCTOR` and click the `OK` button so that `INSTRUCTOR` is visible on top of `SECTION`.

In the Object Navigator, click the icon next to `INSTRUCTOR` to open another Layout Editor window.

e) How can you position the items in the `INSTRUCTOR` block on the `INSTRUCTOR` canvas?

Answer: Select all of the items in the Object Navigator and use the Property Palette to set their `Canvas` *property to* `INSTRUCTOR`.

Note that it makes no difference which instance of the Layout Editor you use to manipulate the items on the `INSTRUCTOR` stacked canvas. You can

manipulate them using the Layout Editor that displays INSTRUCTOR in Stacked Views mode on top of the content canvas, or you can use the instance of the Layout Editor that displays the INSTRUCTOR canvas on its own.

Run the form and issue a query. The Change View button will not work yet, so use Block | Next Block from the Main Menu item to test the stacked canvas. Exit the form and return to the Form Builder.

f) Which properties can you adjust to make the stacked canvas and its items look better? For example, isn't the stacked canvas set a bit lower than the content canvas?

Answer: You can set the Bevel *property to* None *and re-arrange the items and their prompts.*

g) Which of the stacked canvas' properties can you change so that it is not visible when the form opens?

Answer: You can set the INSTRUCTOR *canvas'* Visible *property to* No.

As you noticed for the INSTRUCTOR canvas, if the Visible property is set to Yes and the form has not navigated to an item on a canvas that will cover INSTRUCTOR, then INSTRUCTOR will be visible. The layering of INSTRUCTOR on top of SECTION determines the stacking order of the canvases. Like default navigation, the stacking order of canvases is determined by the order of the canvases in the Object Navigator. Since INSTRUCTOR is listed after the SECTION content canvas, it appears on top. If there had been other canvases after INSTRUCTOR, those would have appeared on top.

■ *FOR EXAMPLE:*

Figure 8.8 shows the Object Navigator with two more stacked canvases added to this form. Note that CANVAS_3 and CANVAS_4 come after SECTION and INSTRUCTOR.

Figure 8.9 shows these same canvases at run-time, just after the form has been opened.

Note that because CANVAS_4 is the last canvas listed in the Navigator, it is displayed on top at run-time. The default stacking order of canvases is similar to default navigation in that it can be overridden programmatically at run-time.

Open the WHEN-BUTTON-PRESSED trigger for the CHANGEVIEW button.

Figure 8.8 ■ Stacking order of canvases in the Object Navigator.

h) What trigger code could you write to change the canvas' `Visible` property? Your trigger should first evaluate whether or not the canvas is visible. If the canvas is not visible, use a built-in to show it. If it is visible, use a built-in to hide it. Use conditional logic and `GET_` and `SET_` built-ins to accomplish this.

Answer: See the discussion below.

The code for the trigger would be as follows:

```
DECLARE
   v_visible VARCHAR2(50);
BEGIN
   v_visible := GET_VIEW_PROPERTY('INSTRUCTOR', VISIBLE);
   IF v_visible = 'TRUE' THEN
      SET_VIEW_PROPERTY('INSTRUCTOR', VISIBLE, PROPERTY_FALSE);
   ELSE
      SET_VIEW_PROPERTY('INSTRUCTOR', VISIBLE, PROPERTY_TRUE);
   END IF;
END;
```

Figure 8.9 ■ Stacked canvases at run-time.

go to contents

The GET_VIEW_PROPERTY statement returns TRUE if the canvas is visible and FALSE if it is not. The IF, THEN, ELSE statement evaluates the current state of the canvas to see whether it should be hidden or displayed.

Run the form, issue a query, and test the Change View button.

i) What are two other built-ins you could use to hide and show the canvas? These should not be GET_ or SET_ built-ins, and they should not cause navigation.

The SHOW_VIEW and HIDE_VIEW built-ins would work just as well and can simply replace the SET_VIEW_PROPERTY statements. The code would be as follows:

```
DECLARE
  v_visible VARCHAR2(50);
BEGIN
  v_visible := GET VIEW PROPERTY('INSTRUCTOR', VISIBLE);
  IF v_visible = 'TRUE' THEN
        HIDE_VIEW('INSTRUCTOR');
    ELSE
        SHOW_VIEW('INSTRUCTOR');
    END IF;
END;
```

Run the form, issue a query, and test the Change View button. Exit the form and return to the Form Builder.

Replace the built-ins you used for Question i with GO_BLOCK('INSTRUCTOR') and GO_BLOCK('ENROLLMENT') statements.

Run the form, issue a query, and test the Change View button.

j) How is the behavior of the form different now that you are using navigational built-ins instead of the built-ins you used in Questions h and i?

Answer: The cursor moves to the first item in the block that is being navigated to.

The GO_BLOCK built-in is similar to GO_ITEM in that it forces navigation. The form automatically navigated to the first item in the block being referenced in the built-in.

LAB 8.3 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions:

1) Which of the following is not true about creating stacked canvases?
- **a)** ____ You cannot position items on a stacked canvas only
- **b)** ____ You can create a stacked canvas visually in the Layout Editor
- **c)** ____ You can create a stacked canvas in the Layout Wizard
- **d)** ____ a & c

2) Where can stacked canvases be positioned?
- **a)** ____ On top of a content canvas
- **b)** ____ Outside the MDI window
- **c)** ____ Behind the MDI window
- **d)** ____ a & b

3) What is a stacked canvas' viewport?
- **a)** ____ The area that is visible at run-time
- **b)** ____ A graphic object used to frame objects
- **c)** ____ The logical container of a canvas
- **d)** ____ a & b

4) Which of the following built-ins can be used to display a stacked canvas?
- **a)** ____ `GO_BLOCK`
- **b)** ____ `SHOW_VIEW`
- **c)** ____ `SET_VIEW_PROPERTY`
- **d)** ____ All of the above

5) What can a stacked canvas be used for?
- **a)** ____ Partly or wholly obscuring other canvases and items
- **b)** ____ Creating wizard objects
- **c)** ____ Dynamically changing the look of the screen
- **d)** ____ All of the above

Quiz answers appear in Appendix A, Section 8.3.

LAB 8.4

TOOLBAR CANVASES

LAB OBJECTIVES

After this Lab, you will be able to:

- Create a Toolbar Canvas
- Use the Toolbar in Another Form

A toolbar canvas properly positions toolbar items in a window. By specifying a canvas as either a vertical or a horizontal toolbar, Forms will position it horizontally along the top of a window or vertically along the left-hand edge of the window.

Toolbars, like all canvases, are made visible by assigning them to windows. Unlike other canvas types, toolbar canvases can be assigned to the MDI window. You will experiment with assigning a toolbar to both a normal window and the MDI window in the Exercises.

Since you are creating a toolbar, you must also create a group of items to place on it. In the Exercises below, you will create buttons. However, it is possible to include other item types on a toolbar. For example, the Utility toolbar in the Layout Editor includes two list items.

In the Exercises, you will create a simple toolbar that mimics some of the functionality of the default toolbar that Forms provides for all forms. Once the toolbar is created and configured, you will learn how to reuse the toolbar by copying it from one form to another.

go to contents

LAB 8.4 EXERCISES

8.4.1 CREATE A TOOLBAR CANVAS

Create a new form and call it `R_TOOLBAR.fmb`. Create a canvas using the Object Navigator and name it `TOOLBAR`. Rename `WINDOW1` to `MAINWIN`.

a) How can you change this canvas into a horizontal toolbar canvas?

This simple toolbar canvas will have five buttons: `SAVE`, `EXIT`, `ENTER_QUERY`, `EXECUTE_QUERY`, and `CANCEL_QUERY`.

b) What type of object will you have to create to logically contain these items?

Create a block and name it `TOOLBAR`. Create five buttons. Name them `SAVE`, `EXIT`, `ENTER_QUERY`, `EXECUTE_QUERY`, and `CANCEL_QUERY`. They should appear in this order in the block. Assign the buttons to the `TOOLBAR` canvas.

c) Do typical toolbars have text labels or iconic labels? Which property can you adjust to make your toolbar buttons typical?

Use the icons supplied by Oracle Developer. The item names are in CAPS and the corresponding icon names are in lower-case as follows: `SAVE save`, `EXIT exit`, `ENTER_QUERY entqry`, `EXECUTE_QUERY exeqry`, and `CANCEL_QUERY canqry`.

go to contents

d) Should the user be able to navigate to these buttons with the mouse or keyboard? Which properties should you set to ensure that they can't?

Adjust the size properties of the buttons and set `Height` to `19` and `Width` to `19`. If you haven't done so already, open the Layout Editor to view the `TOOLBAR` canvas.

e) What Layout Editor feature can you use to quickly position all of the buttons next to each other? Note that if you used the Object Navigator to create the buttons, they may be currently positioned on top of each other. This should not pose a problem for you.

Adjust the `Height` of the `TOOLBAR` canvas to `21` to make the buttons fit better.

f) What property can you use to give each button a Tool Tip?

Temporarily set the `SAVE` button's `Keyboard Navigable` property to `Yes`. This will allow you to run the form and view the toolbar without having to create any other blocks or navigable items. You will switch it back later.

Run the form.

g) Which object is the toolbar attached to? Which of the `TOOLBAR` canvas' properties indicate this?

h) Can you assign the toolbar to the module? How do you think this will affect the position of the toolbar?

Assign the TOOLBAR to the module and run the form again.

i) Where is the toolbar now? Is the original default toolbar still there?

Change the module's Menu Module property from DEFAULT&SMARTBAR to DEFAULT. Run the form.

Create a WHEN-BUTTON-PRESSED trigger for the EXIT button with the following code:

```
DO_KEY('EXIT_FORM')
```

j) What happened to the default toolbar?

Use the wizards to quickly create a block based on the ZIPCODE table. Include all of the items in the block. Enforce data integrity should be **unchecked**. Stop when you get to the Layout Wizard's canvas page.

k) Should you position the ZIPCODE items on the TOOLBAR canvas? If not, what type of canvas should you create for them?

Continue creating the canvas for the ZIPCODE block. When you are finished, change the SAVE button's Keyboard Navigable property back to No. Run the form and test that the toolbar appears properly.

l) What built-ins can you use to respond to the rest of the buttons? Create a `WHEN-BUTTON-PRESSED` trigger for each button and use `DO_KEY` statements with the built-ins.

m) Does it make sense to have the `SAVE` button enabled during Enter Query mode? Why not?

n) What built-in could you use to adjust this at run-time?

Save the form as R_TOOLBAR.fmb.

8.4.2 USE THE TOOLBAR IN ANOTHER FORM

In this Exercise, you will copy the toolbar and its objects to the `R_CONCANV.fmb` form. But, before you do, you will add another button to the toolbar. Open both `R_TOOLBAR.fmb` and `R_CONCANV.fmb` in the Form Builder.

a) What item on the `R_CONCANV.fmb` form could be better implemented as a toolbar item?

Minimize all of the objects for `R_CONCANV.fmb` for the time being. Create another button on the `TOOLBAR` canvas in `R_TOOLBAR.fmb` and position it to the right of `Cancel Query`. Name it `SHOW_HIST`. Adjust its size, navigable, and icon properties to match the other buttons on the `TOOLBAR` canvas. Set its `Icon Filename` property to `srch_frw` and set its Tool Tip to `Record History`.

go to contents

This button will be used to open the HISTWIN window in the R_CONCANV.fmb form.

b) What code should you put in the WHEN-BUTTON-PRESSED trigger for the SHOW-HIST button?

Add the code and compile the trigger.

c) Which objects should you drag from R_TOOLBAR.fmb to R_CONCANV.fmb so that it too can have a functioning toolbar? Copy the objects; do not subclass them.

d) What should you do to make this toolbar available to the form? What else should you do to make it the only toolbar in the form?

Run the form and test all of the toolbar buttons.

e) Was it convenient to have to drag these objects one at a time into R_CONCANV.fmb?

Save the changes to R_TOOLBAR.fmb. *Do NOT save the changes to* R_CONCANV.fmb. *If you already have, simply delete all of the* TOOLBAR *objects from* R_CONCANV.fmb *and re-save it.*

LAB 8.4 EXERCISE ANSWERS

8.4.1 ANSWERS

Create a new form and call it `R_TOOLBAR.fmb`. Create a canvas using the Object Navigator and name it `TOOLBAR`. Rename `WINDOW1` to `MAINWIN`.

a) How can you change this canvas into a horizontal toolbar canvas?

Answer: Change its `Canvas Type` *property to* `Horizontal Toolbar`.

b) What type of object will you have to create to logically contain these items?

Answer: You will have to create a non-base table block.

A toolbar is very similar to the forms you created for data entry and querying. It requires items that must be positioned on a canvas and contained in a block. The main difference is that there are no data items on a toolbar, only non-data items.

c) Do typical toolbars have text labels or iconic labels? Which property can you adjust to make your toolbar buttons typical?

Answer: Typical toolbars have iconic labels. You should adjust the `Iconic` *property.*

d) Should the user be able to navigate to these buttons with the mouse or keyboard? Which properties should you set to ensure that they can't?

Answer: No, set both `Keyboard Navigable` *and* `Mouse Navigable` *to* `No`.

In certain cases, you have seen that it is often necessary for a user to be able to navigate, at least with the keyboard, to a button. This is especially true if they are doing data entry and would like to use the ENTER key to initiate the Button Pressed event. However, it is never necessary to navigate to a button on a toolbar.

e) What Layout Editor feature can you use to quickly position all of the buttons next to each other? Note that if you used the Object Navigator to create the buttons, they may be currently positioned on top of each other. This should not pose a problem for you.

Answer: You can use `Arrange | Align Objects`, `Stack Horizontally`.

Stacking the buttons will put them next to each other as shown in Figure 8.10.

In many applications, you may have noticed that the toolbar buttons are spaced or separated into groups to make the toolbar more understandable.

go to contents

Figure 8.10 ■ A toolbar with buttons stacked horizontally.

■ *FOR EXAMPLE:*

You may wish to organize your toolbar so that the buttons are spaced by function. You could have two groups, one for the `SAVE` and `EXIT` buttons, and one for the `QUERY` buttons. The groups could be separated by blank space and even graphic objects as in Figure 8.11.

The graphic object is a line with its `Bevel` property set to `Inset`.

f) What property can you use to give each button a Tool Tip?

Answer: You can use the `Tool Tips` *property.*

While the button icons are usually quite descriptive, it is always helpful to include Tool Tips, especially for new users.

Run the form.

g) Which object is the toolbar attached to? Which of the `TOOLBAR` canvas' properties indicate this?

Answer: The `TOOLBAR` *canvas is attached to* `MAINWIN`*. The canvas'* `Window` *property indicates this.*

As you know, all canvases must be assigned to a window to be visible. The `TOOLBAR` canvas was attached by default to `MAINWIN` when you created it. Interestingly, all windows have a `Horizontal Toolbar` and `Vertical Toolbar` property. So, you could assign the toolbar canvas at the window level as well.

There are no other canvases in this form yet, but if there were, how do you think `TOOLBAR` would be positioned relative to those canvases?

■ *FOR EXAMPLE:*

Assume you have created a block based on the `STUDENT` table and have positioned all of its items on a content canvas called `STUDENT`. Both the `STUDENT` and `TOOLBAR` canvases are assigned to `MAINWIN`. How will the

Figure 8.11 ■ A toolbar with buttons separated by function.

go to contents

TOOLBAR canvas behave? Will it stack itself on top of the STUDENT content canvas or will it position itself between the window's title bar and the STUDENT content canvas?

The TOOLBAR will position itself between the window's title bar and the STUDENT canvas. It will not behave like a stacked canvas and position itself on top of the STUDENT canvas. You don't have to be concerned about toolbars, be they horizontal or vertical, obscuring the content canvas.

In the next question, this will become a moot point because you will learn how to attach your custom toolbar to the MDI window.

h) Can you assign the toolbar to the module? How do you think this will affect the position of the toolbar?

Answer: Yes, using the form's `Horizontal Toolbar` *property. It will assign the* TOOLBAR *canvas to the MDI window.*

Assign TOOLBAR to the module and run the form again.

i) Where is the toolbar now? Is the original default toolbar still there?

Answer: It is attached to the MDI window. Yes, the original, default toolbar is still there.

Multiple horizontal toolbars are extremely common. Applications like MS Word can have as many as six horizontal toolbars. Even the Layout Editor has two horizontal toolbars. In this case, the toolbars have functionality in common, so it will not be necessary to have both.

j) What happened to the default toolbar?

Answer: The default toolbar has disappeared.

The TOOLBAR you have created is now the main toolbar for the application.

The `&SMARTBAR` value in the `Menu Module` property assigns a menu toolbar, also known as a smart bar, to the default Forms menu. The default menu and smart bar are objects that Forms attaches to every form at runtime by default. As you can see, you can override these default settings and use your own toolbars. In Chapter 13, "Forms Menus," you will learn how to override the defaults to create your own menus.

Use the wizards to quickly create a block based on the ZIPCODE table. Stop when you get to the Layout Wizard's canvas page.

k) Should you position the ZIPCODE items on the TOOLBAR canvas? If not, what type of canvas should you create for them?

Answer: No, you should not position them on the TOOLBAR *canvas. You should create a new content canvas for them.*

l) What built-ins can you use to respond to the rest of the buttons? Create a WHEN-BUTTON-PRESSED trigger for each button and use DO_KEY statements with the built-ins.

Answer: See the discussion below.

Use the following statements for each button:

```
1) SAVE - DO_KEY('COMMIT_FORM');
2) EXIT - DO_KEY('EXIT_FORM');
3) ENTER_QUERY - DO_KEY('ENTER_QUERY');
4) EXECUTE_QUERY - DO_KEY('EXECUTE_QUERY');
5) CANCEL_QUERY - DO_KEY('EXIT_FORM');
```


Note that the CANCEL_QUERY button also uses the EXIT_FORM built-in because EXIT_FORM switches the form from Enter Query mode back to Normal mode.

m) Does it make sense to have the SAVE button enabled during Enter Query mode? Why not?

Answer: No, because you cannot commit records during ENTER_QUERY *mode.*

In fact, there are two other buttons that should not be enabled in Enter Query mode. The EXIT_FORM button should not be enabled because you should cancel the query before being able to exit the form. The ENTER_QUERY button should not be enabled either because you are in Enter Query mode already, which means there is no need to click the button again.

By the same token, the CANCEL_QUERY button should be disabled when the form is not in Enter Query mode.

Disabling and enabling buttons on a toolbar make an application much more user-friendly. The status of the buttons tells a user what she can and can't do at any given time.

Figure 8.12 ■ The toolbar in Enter Query mode.

go to contents

Figure 8.13 ■ The toolbar in Normal mode.

n) What built-in could you use to adjust this at run-time?

Answer: You could use the `SET_ITEM_PROPERTY`.

In Question m, you learned that many buttons will have to be enabled and disabled depending on the state of the form. In Enter Query mode, the toolbar should look like Figure 8.12. In Normal mode, it should look like Figure 8.13.

You could add a series of `SET_ITEM_PROPERTY` statements to your triggers to enable and disable the buttons as the status of the form changes, but this would be a bit messy. You want to write the PL/SQL so that the `SET_ITEM_PROPERTY` statements can be reused instead of rewritten over and over again. In the "Test Your Thinking" section of Chapter 10, "Reusable Code," you will edit the `WHEN-BUTTON-PRESSED` triggers to call a program unit that will enable and disable certain buttons depending on the system mode.

Save the form as `R_TOOLBAR.fmb`.

8.4.2 ANSWERS

a) What item on the `R_CONCANV.fmb` form could be better implemented as a toolbar item?

Answer: The `SHOW_HIST` *button.*

Remember that eventually you will want both the `TOOLBAR` canvas and `HISTORY` canvas on all of your forms. Therefore, it makes sense to add the `SHOW_HIST` button to the toolbar.

This button will be used to open the `HISTWIN` window in the `R_CONCANV.fmb` form.

b) What code should you put in the `WHEN-BUTTON-PRESSED` trigger for the `SHOW-HIST` button?

Answer: See the discussion below.

You will use the GO_ITEM built-in and add it to the trigger as shown below:

```
:global.last_item := :SYSTEM.CURSOR_ITEM;
GO_ITEM('CONTROL.HIDE_HIST');
```

Add the code and compile the trigger.

c) Which objects should you drag from R_TOOLBAR.fmb to R_CONCANV.fmb so that it too can have a functioning toolbar? Copy the objects; do not subclass them.

Answer: See the discussion below.

You should copy the following objects:

1) The TOOLBAR block.
2) The TOOLBAR canvas.

These two objects work together to make the toolbar visible and functional.

d) What should you do to make this toolbar available to the form? What else should you do to make it the only toolbar in the form?

Answer: Set the module's Horizontal Toolbar *property to* TOOLBAR. *Remove* &SMARTBAR *from the module's* Menu Module *property.*

Run the form and test all of the toolbar buttons.

e) Was it convenient to have to drag these objects one at a time into R_CONCANV.fmb?

Answer: Not really.

You have created the toolbar to be a reusable object, but reusing it is a bit of a nuisance. In the next Chapter, you will create two more objects to include as part of the toolbar, so reusing it as you have here would become even more tedious. In the next Chapter, you will learn how to package the toolbar objects together so that you can move them as a single entity from form to form quickly and easily.

Save the changes to R_TOOLBAR.fmb. *Do NOT save the changes to* R_CONCANV.fmb. *If you already have, simply delete all of the* TOOLBAR *objects and re-save it.*

LAB 8.4 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions:

1) Which of the following is not true about toolbars?
- **a)** ____ They can be attached to the MDI window
- **b)** ____ They can be positioned vertically or horizontally
- **c)** ____ They can be attached to an LOV
- **d)** ____ They can have list items as objects

2) How can you completely hide a toolbar button?
- **a)** ____ Set its `Canvas` property to `Null`
- **b)** ____ Set its `Visible` property to `No`
- **c)** ____ Set its `Window` property to `WINDOW1`
- **d)** ____ a & b

3) Which of the following is true about toolbar buttons?
- **a)** ____ They can be enabled or disabled at run-time
- **b)** ____ They can be iconic
- **c)** ____ They can have `WHEN-BUTTON-PRESSED` triggers attached to them
- **d)** ____ All of the above

4) At what level in the Forms hierarchy can the triggers that respond to toolbar item events be?
- **a)** ____ Item level
- **b)** ____ Block level
- **c)** ____ Form level
- **d)** ____ All of the above

Quiz answers appear in Appendix A, Section 8.4.

CHAPTER 8

TEST YOUR THINKING

1) Create four new iconic buttons on the toolbar in `R_TOOLBAR.fmb` for the following functions:

- New record
- Delete record
- Previous record
- Next record

Identify the built-ins to correspond to these buttons and add the code to the `WHEN-BUTTON-PRESSED` trigger. Save the form as `R_TOOLBAR.fmb`.

2) Implement the toolbar in `R_TOOLBAR` as a vertical toolbar. Save this form as `R_TOOLBAR_V.fmb`.

3) Use the wizards to create a form with tabbed canvases. This was not covered in the Lab sections of this Chapter, but you should be able to do it based on your knowledge of canvases. The form will be a master-detail-detail form based on the `INSTRUCTOR`, `SECTION`, and `ENROLLMENT` tables. The items from each block should appear on a different tab page. Arrange and align the items as you see fit.

4) Create a scrolling stacked canvas. First you must do some minor setup. Create a master-detail form based on the `COURSE` and `SECTION` tables. Include the audit columns in the blocks, but do not display them on the canvas. Add a display item called `INSTRUCTOR_NAME` to the `SECTION` block and populate it with a `POST-QUERY` trigger.

IMPORTANT: First use the wizards to position all of the items on a content canvas, as you would if you were creating a simple master-detail form. Do not try to create a stacked canvas using the wizard.

Once you have finished with the wizard and the items are positioned on the content canvas, use the Layout Editor to create a stacked canvas. Then, reassign some of the items in the `SECTION` block to this canvas. It should look like Figure 8.14. You will have to move the items to this canvas by adjusting their properties.

go to contents

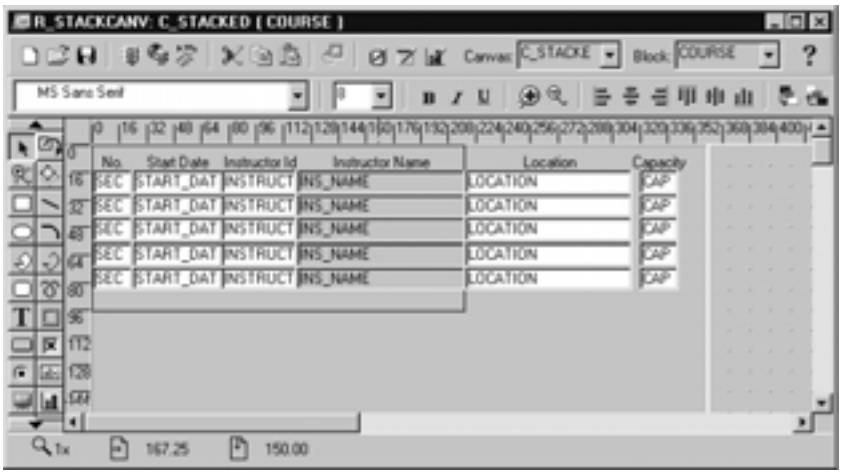

Figure 8.14 ■ Stacked canvas with some of the items from the `SECTION` block.

Note that, while the items are on the canvas, not all of them are within view. This is very important to simulate the scrolling effect. Also note that there is one item from the `SECTION` block that is not on the stacked canvas. Which item is it? Which canvas do you think it is on?

Use the Stacked Views feature to align and arrange the items. When the form is running, it should look like Figure 8.15.

Figure 8.15 ■ A running form with a scrolling stacked canvas.

go to contents