

1. Developing Forms



Module Objectives

In this section, we are going to take a more in-depth look at Visual FoxPro forms by learning some new terms, as well as, applying ***some*** of the terminology presented in the previous section. You will learn about:

- ?? The Data Environment
- ?? Container Classes
- ?? Forms Sets
- ?? Referencing Form Objects
- ?? The _SCREEN Object

Data Environment

Overview

The most common reason to build forms is to display and update the data in the application. Before you start building forms, you have to understand how forms work with data. Forms use the Data Environment to integrate data into a form.

The Data Environment contains information about all the tables, relationships, and views, used by a form or report. When you save a form or report, the Data Environment information is saved with it. The Data Environment is modified in the Data Environment Designer.



When the Data Environment of a form or report is defined, specifying . T . in the AutoOpenTables and AutoCloseTable properties instructs Visual FoxPro to open the tables or views when you open or run the file and close them when the file is closed or released.

Accessing the Data Environment

There are several ways to access the Data Environment.

??  **Data Environment:** Click on the Data Environment button on the Forms Designer toolbar.

Or

?? Select the **View | Data Environment** menu option.

Or

?? Select Data Environment from the shortcut menu

Adding Tables to the Data Environment

There are a couple of ways to add tables to the Data Environment.

?? Select the **DataEnvironment | Add** menu option

Or

?? Select Add from the shortcut menu

Adding a Bound Text Box Control from the Data Environment

A bound text box can be quickly created by clicking and dragging a field from the Data Environment and placing it on the form (A *bound control* is a control that is directly tied to the data in a specific field).

Adding Multiple Bound Fields from the Data Environment

You can add multiple bound controls quickly by selecting the fields to be placed from the Data Environment. Right click on one of the fields and drag it on to the form. Visual FoxPro will display a shortcut menu asking if you want a grid or multiple controls for the fields you selected.



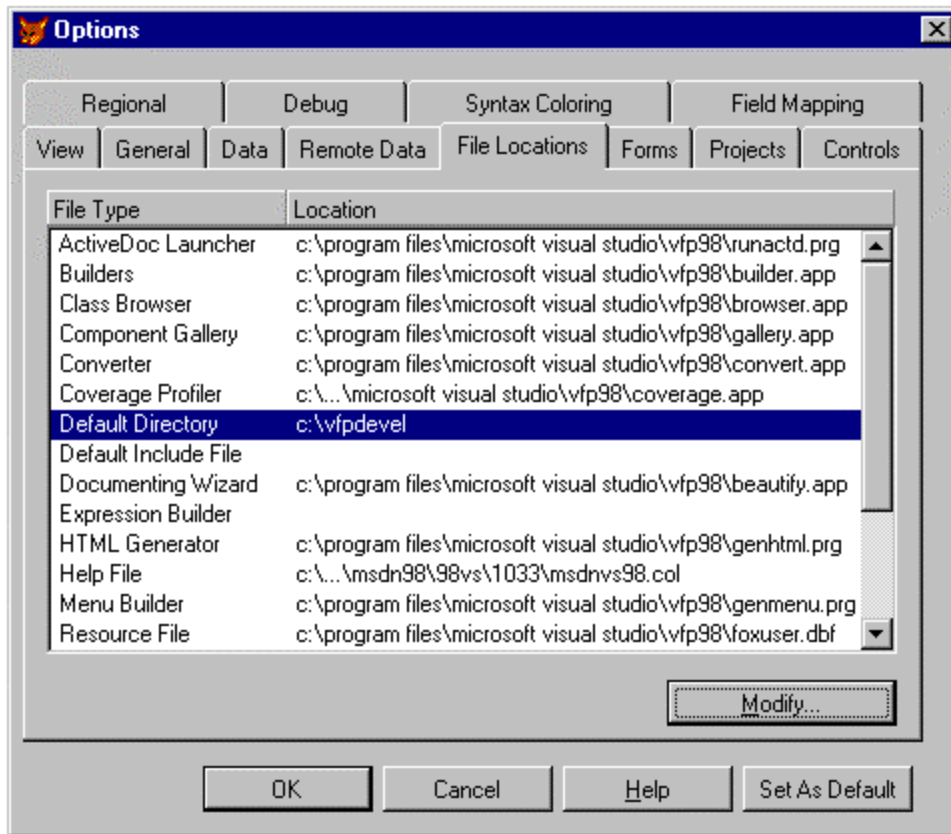
Adding a Bound Grid Control from the Data Environment

A bound grid control can be quickly created by clicking and dragging an entire table from the Data Environment and placing it on the form.

Student Exercise: Add Bound Controls to a Form With the Data Environment

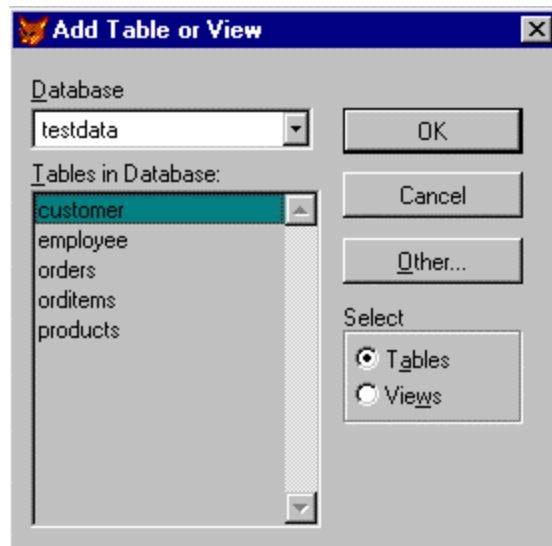
In this exercise, we will add bound text boxes from the Data Environment to a form. First though, we need to prepare the Visual FoxPro environment to suit our training class needs.

1. Select the **Tools | Options** menu option to access the Options dialog. Select File Locations.



2. Click the **Default Directory** check box and enter **C : \VFPDEVEL** as the default directory. Click **Set as Default** to save this setting for future Visual FoxPro sessions. Click **OK**.
3. Create a new project named **VFPDevel**.
4. Add the **TestData** database to the project. We'll be using the sample data for our exercises. (TestData.DBC ships with Visual FoxPro and can be found in the **\Program Files\Microsoft Visual Studio\MSDN98\98VS\1033\SAMPLES\VFP98\DATA** subdirectory).
5. Select the **Documents** tab, select **Forms** and click **New**.
6. Select **New Form** and click **OK**.

7. **Right-click on the form** to open the shortcut menu and select **Data Environment**.
8. The Add View dialog will appear.



9. Select the **Employee** table and click **Add**. Click **Close** to return to the Data Environment.
10. Size the Data Environment window so that you can see the Employee table.
11. Click on the **Emp_id** field, drag it over to the form and release the mouse to place a *bound* control. What is the Caption property of the label created next to the textbox? What if you wanted Employee ID instead of Emp_id to be used automatically? Where would you make that change?
12. Individually place the **Last_name**, **First_name** and **Title** fields on the form using the same technique.
13. Now add the **Address**, **City** and **Region** fields by selecting them and then dragging one of them on to the form with the right mouse button.
14. Select Multiple Controls from the shortcut menu displayed.



15. Add a command button and set the following properties:

Property	Value
Caption	Done
Name	CmdDone

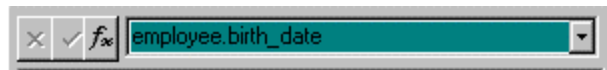
16. In the Click event of the button type **RELEASE THISFORM**.
17. **Save** the form as **frmEmpInfo**.

18. Run the form by clicking the Run tool  in the Standard toolbar.
19. Close the form.

Student Exercise: Adding Bound Controls Manually

In this exercise, we will add a bound control manually by making the appropriate entries in the Property window.

1. If it isn't already in the Form Designer, modify the **frmEmpInfo** form.
2. Add a text box to the form.
3. Select the **Control Source** property (its on the Data tab).



4. Select the drop-down list in the Property Settings Box and select the **birth_date** field.
5. **Save** and **run** the form.
6. Close the form.

The Data Environment

Overview

In the previous section, you built a form and were introduced to the Data Environment. You added a table to the Data Environment and build a form with bound controls on it. In this section we'll look a little deeper at the Data Environment and what you need to know to work with it effectively.

In OOP terms, the Data Environment is a container object that contains Cursor and Relation objects. Data Environment properties can only be accessed while the Data Environment Designer is open.

Cursor Objects

When tables or views are added to the Data Environment, a cursor object is created for each table. A *cursor* (**CUR**rent **S**et **O**f **R**ecords) is a representation of a set of records.

For example, if a form accesses a single table, there would only be one cursor object in the Data Environment. If a form accesses two tables there would be two cursor objects in the Data Environment.

Relation Objects

When you have more than one table in the Data Environment, the Relation object allows you to specify characteristics about how those tables are related.

Commonly Used DATA ENVIRONMENT Object Properties

Property	Description
AutoCloseTables	Specifies if the tables should be closed when the form closes.
AutoOpenTables	Specifies if the tables should be opened when the form loads.
InitialSelectedAlias	Specifies which work area should be selected when the form loads.

Commonly Used DATA ENVIRONMENT Object Events

Event	Description
AfterCloseTables	Occurs after tables are automatically closed.
BeforeOpenTables	Occurs before tables are automatically opened.

Commonly Used DATA ENVIRONMENT Object Methods

Method	Description
--------	-------------

CloseTables	Causes the tables and views to be closed.
OpenTables	Causes the tables and views to be opened.

Commonly Used CURSOR Object Properties

Property	Description
Alias	Specifies the name of the table or view referenced in the cursor.
BufferModeOverride	Specifies whether or not to override the buffer setting for the form.
Exclusive	Specifies if the table should be open exclusively.
Filter	Specifies a filtering criteria to exclude records from the cursor.
Name	Specifies the name of the cursor.
Order	Specifies the index tag to use initially.
ReadOnly	Specifies if the cursor is read-only.

Commonly Used RELATION Object Properties

Property	Description
ChildAlias	Specifies the name of the child table in the relationship.
ChildOrder	Specifies the index tag to use for the child.
Name	Specifies the name of the relation object.
OneToMany	Specifies if a one-to-many relation exists between the two tables.
RelationalExpr	Specifies the expression used to join the two tables.

Container Object

Overview

Container objects contain other objects and allow access to the objects contained within them. For example, forms are containers. They contain the controls you place on them.

The following table lists what each container class can contain.

Container	Contains
Command Button Groups	Command Buttons
Data Environment	Cursors and Relations
Form Sets	Forms and Toolbars
Forms	Controls
Grids	Columns
Grid Columns	Controls
Option Button Group	Option Buttons
Page Frames	Pages
Pages	Controls
Toolbars	Controls

.Parent Object Reference

The Parent object reference references the immediate container object of a control. For example, suppose there is a textbox control named txtControl on a page named pagSales on a page frame named pgfStuff. From txtControl, we could change the background color of page pagSales with:

THIS.Parent.BackColor = RGB(255,0,0)

The following would hide the entire set of pages all together:

THIS.Parent.Parent.Visible = .F.

There will be more on object referencing in a later section.

Collections and Counts

Every container object has an associated collection property which stores a reference to each item in the container. A Collection is just a fancy term for an array. If you understand arrays, you understand collections.

For example, a form is a container that contains controls. Every form object has a collection property Controls that stores a reference to each control on the form.

The number of items in the container is stored in a *count* property. For example, the ControlCount property stores the number of controls on a form and the PageCount property stores the number of pages objects in a page frame.

The following table lists the name of each container and its associated collection and count property names.

Container	Collection Name	Count Name
Command Button Groups	Buttons	ButtonCount
Form Sets	Forms	FormCount
Forms	Controls	ControlCount
Grid Columns	Controls	ControlCount
Grids	Columns	ColumnCount
Option Button Groups	Buttons	ButtonCount
Page Frames	Pages	PageCount
Pages	Controls	ControlCount
Toolbars	Controls	ControlCount

Accessing Objects in a Collection

The following code demonstrates a simple way to disable all textboxes on a form without knowing their name. The code will loop from 1 to the number of controls on the form. Each control in the forms Controls collection will be interrogated to see if the control is a textbox. If it is a textbox, then its Enabled property will be set to false. This code will work if there are 2 or 200 controls on a form.

```
FOR i = 1 TO THISFORM.ControlCount
    IF THISFORM.Controls(i).BaseClass = "textbox"
        THISFORM.Controls(i).Enabled = .F.
    ENDIF
ENDFOR
```

WITH...ENDWITH

The WITH . . . ENDWITH clause is used to allow multiple statements to be run on a single object without having to specify the name of the object.

Syntax:

```
WITH ObjectName
    [.cStatements]
ENDWITH
```

The *ObjectName* specifies the name of the object that multiple properties will be set for.

cStatements consist of any number of commands that are used to set properties for ObjectName. Properties are set by placing a period before each property and assigning a value.

Example:

Here's the same code from above using WITH...ENDWITH.

```
WITH THISFORM
```

```
    FOR i = 1 TO .ControlCount
```

```
        IF .Controls(i).BaseClass = "textbox"
```

```
            .Controls(i).Enabled = .F.
```

```
        ENDIF
```

```
    ENDFOR
```

```
ENDWITH
```

Tip: WITH ...ENDWITH is a HUGE timesaver. It makes code easier to write and reduces code clutter, therefore making code easier to read. You will see it used very often in class.

Form Sets

Overview

If a form is just one form object, then a form set must be multiple form objects. Correct! You can manipulate multiple forms as a group by including them in a form set. A *form set* is a container for one or more forms.

Form sets have the following benefits:

- ?? You can show or hide all the forms in a form set at one time.
- ?? You can visually arrange multiple forms at once to control their relative positions.
- ?? With a data environment created at the forms set level, you can automatically synchronize record pointers in multiple forms. If you change the record pointer in a parent table in one form, the child records in another form are updated and displayed.

Creating a Form Set

By default a new form is not a form set. You need to specify that you want to work with a form set. This is done by selecting the **Form | Create Form Set** menu option. It won't look like much happened after you select **Form | Create Form Set**. What actually happened is that the **Form | Add Form** menu option has been activated.

Adding Forms to a Form Set

Once a Form set is created, forms are added by selecting the **Form | Add New Form** menu option. A second (or third, or fourth, or fifth...) form will appear. The forms are individual objects with properties and must be treated as such.

Removing a Form From a Form Set

You Remove a form from a form set by selecting the **Form | Remove Form** menu option.

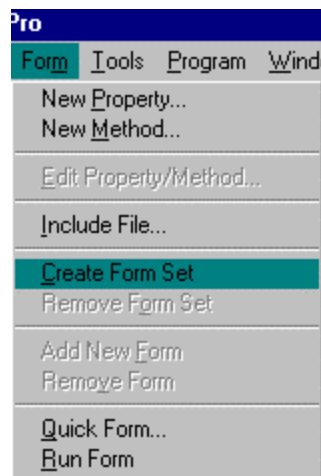
Closing a Form Set at Runtime

The `RELEASE THISFORMSET` command releases closes the active form set.

Student Exercise: Create a Form Set

In this exercise, we will create a form set and view different information from one table on two forms.

1. Create a new, blank form.
2. Save the form as **frmEmpInfoFormSet**.
3. Open the Data Environment and add the **Employee** table.
4. Drag **Emp_id**, **Last_name** and **First_name** to the form.
5. Click on the form.
6. Select the **Form | Create Form Set** menu option.



7. Select the **Form | Add New Form** menu option.
8. A second form appears in the Form Designer. Notice the first one is called **Form1** and the second is called **Form2**. They are two separate objects.
9. With Form2 in the front, select **Data Environment** from the shortcut menu.
10. Drag the **Address**, **City**, and **PostalCode** fields onto **Form2**.
11. **Save** and **run** the form.
12. Click back and forth on the two screens. How were they arranged when they first came up?
13. Close both forms.

14. What a pain! Having to close both forms individually like that. Reopen the **frmEmpInfoFormSet** form. Place a command button on Form2 and set the following properties:

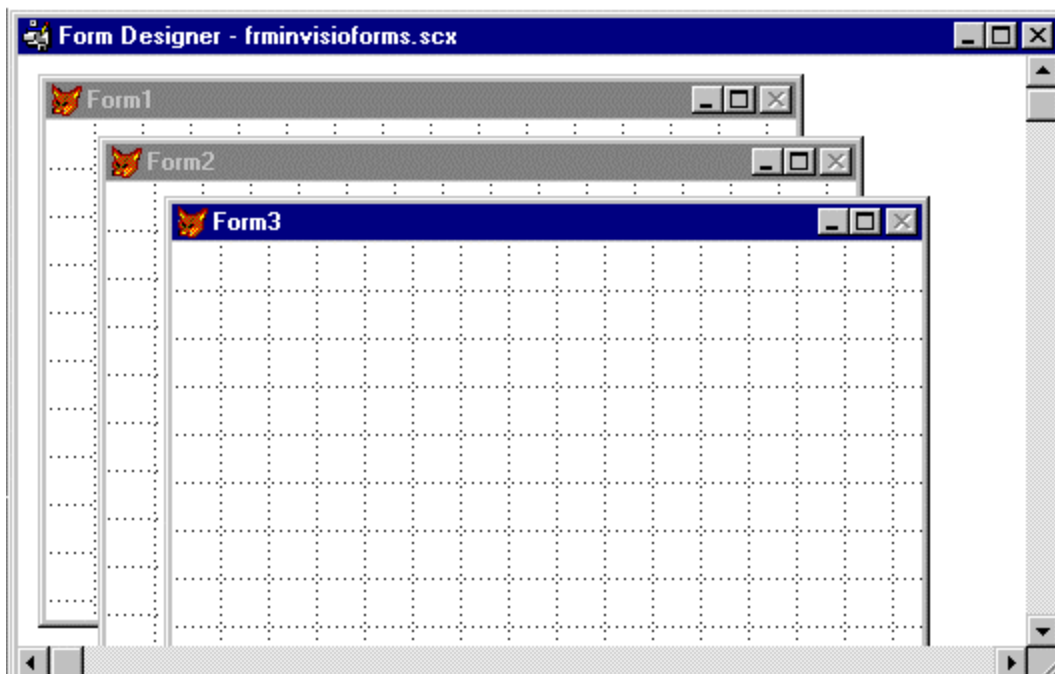
Property	Value
Caption	Done
Name	cmdDone

15. In the **Click event** of the **cmdDone** command button, type **RELEASE THISFORMSET .**
16. Save and run the form.
17. What happens when you click the Done button?

Student Exercise: Fun With Form Sets

In this exercise, we will create a form set that contains three forms. Only one will ever be visible at a runtime though.

1. Create a new, blank form.
2. Save the form as **frsInvisoForms**.
3. Create a form set.
4. Add two more forms to the current form.
5. You should now have **Form1**, **Form2** and **Form3** in the Form Designer (You might have to move the forms to see them all).



6. For **Form2** and **Form3** set the following property:

Property	Value
Visible	.F.

7. Add a command button to **Form1** and set the following properties.

Property	Value
Caption	\<Switch
Name	CmdSwitch

8. In the **Click event** of the **cmdSwitch** button, enter the following (minus the line numbers):

```
THIS.PARENT.Visible = .F.  
THISFORMSET.Form2.Visible = .T.
```

9. Copy cmdSwitch to Form2 and Form3.

10. Modify the Click event of cmdSwitch on Form2 to look like the following code:

```
THIS.PARENT.Visible = .F.  
THISFORMSET.Form3.Visible = .T.
```

11. Modify the **Click event** of **cmdSwitch** on **Form3** to look like the following code:

```
THIS.PARENT.Visible = .F.  
THISFORMSET.Form1.Visible = .T.
```

12. **Save** and **run** the form.

13. Close down the form.

14. Try to modify **frsInvisoForms**. Did you get a “File in Use” message? Why?

15. Type **CLEAR ALL** in the command window to remove the form set from memory.

16. Modify **frsInvisoForms**.

17. Add a command button to **Form3** with the following properties:

Property	Value
Caption	\<Done
Name	CmdDone

18. In the **cmdDone** Click event, enter **RELEASE THISFORMSET**.

19. Save and run the form.

20. Would the following code work behind the Switch button on Form1?

```
THIS.PARENT.Visible = .F.  
THISFORMSET.Forms(2).Visible = .T.
```

Referencing Form Objects

Overview

Referencing objects on the form requires a special naming syntax. When an object exists in a container, you must identify it in relation to the container hierarchy. Items are referenced by the largest object they exist in, down through any subcontainers, and finally down to the control.

For example, on a form named `frmTestForm`, there exist two controls, a command button named `cmdSetValue` and a text box named `txtnAmount`. We want to assign a value of 7 to `txtnAmount` whenever the user clicked on `cmdSetValue`. Here's the code that would go in the `cmdSetValue` click event:

```
frmTestForm.txtnAmount.Value = 7
```

That's the Absolute Referencing Technique for referencing `txtnAmount`. Is `txtnAmount` in a container? Yes.

What container is `txtnAmount` in? The form.

What if we didn't know the name of the form? We could use the Relative Referencing Technique to accomplish the same thing. Relative Referencing is accomplished by using the `THISFORM` prefix. Here's the code using Relative Referencing:

```
THISFORM.txtnAmount.Value = 7
```

Tip: A very common mistake made by Visual FoxPro programmers is to forget to include the `.Value` reference when assigning a value to an object.

Wrong: `THISFORM.txtnAmount = 7`

Right: `THISFORM.txtnAmount.Value = 7`

Student Exercise: Referencing Form Objects

In this exercise, we will create the form we just read about and implement what we've learned:

1. Create a new, blank form.
2. Set the following properties for the form.

Property	Value
AutoCenter	.T.
Caption	Set a Value
Name	FrmTestForm

3. Save the form as **frmTestForm**.
4. Add a command button that will close down the form. (Hint: `RELEASE THISFORM`)
5. Add a command button named with the following properties:

Property	Value
AutoSize	.T.
Caption	Click to Set Value
Name	CmdSetValue

6. Add a text box with the following properties:

Property	Value
Name	txtnAmount
Value	33

7. **Double-click** on the **cmdSetValue** command button and enter the following command in the Click Procedure:

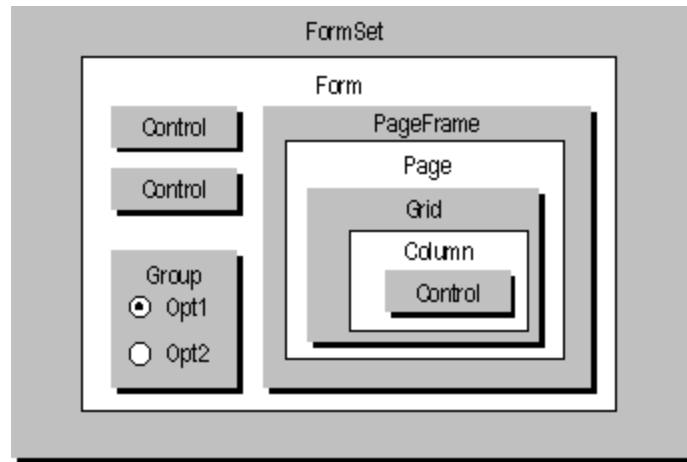
```
THISFORM.txtnAmount.Value = 7
```

8. Save and test the form.

Advanced Object Referencing

Overview

We just saw that when a control is contained within a form, we must reference the form name first, then the control. What if there was a text box control named txtSalary on a page in a page frame on a form. Or even worse, what if txtSalary is contained in a column in a grid, the grid is on a page in a page frame, the page frame is on a form, and the form is a member of a form set. The picture below illustrates that scenario.



To reference the txtSalary control, it's entire container hierarchy must be specified.

Formset . Form . PageFrame . Page . Grid . Column . txtSalary

The following table has a few examples of THIS, THISFORM, THISFORMSET and Parent to set object properties.

Command	Where to put the command
THIS.Caption = "OK"	In the method code of the control whose caption you want to change.
THISFORM.cmd1.Caption = "OK"	In the method code of any control on the same form that cmd1 is on.
THISFORMSET.frm1.cmd1.Caption = "OK"	In the method code of any controls on any form in the form set.
THIS.Parent.BackColor = RGB(0,0,255)	In the method code of a control on a form.

The `_SCREEN` Object

Description

The `_SCREEN` object specifies properties and methods for the main Visual FoxPro window.

ActiveForm Property

The `ActiveForm` property of the `_SCREEN` object allows you to access and manipulate the active form even if you don't know its name.

For example, if we didn't want to see the form that is currently active, we could use:

```
_SCREEN.ActiveForm.Visible = .F.
```

ActiveControl Property

We can manipulate the `ActiveControl` property of the `_SCREEN` object to work with the active control on the active form. For example, to determine the name of the active control, we could use:

```
_SCREEN.ActiveForm.ActiveControl.Name
```

Caption Property

The `Caption` property of the `_SCREEN` object sets the caption for the main Visual FoxPro window. For example, to change the caption for the Visual FoxPro desktop window we could use:

```
_SCREEN.Caption = "Our Application"
```

Student Exercise: Manipulate the Active Form

In this exercise, we will see how the `_SCREEN` object variable works at runtime.

1. Click on the Command window.
2. Enter `_SCREEN.Caption = "Hello World"` in the command window and press Enter.
3. Enter `_SCREEN.BackColor = GETCOLOR()` in the command window and press Enter.
4. Enter `_SCREEN.Left = 100` in the command window and press Enter.
5. Enter `_SCREEN.Top = 50` in the command window and press Enter.
6. Maximize the screen.

For more information on the `_SCREEN` object, see Visual FoxPro Help.

Module Review

In this section, we took a look at forms and techniques associated with running forms. You learned about:

- ?? The Data Environment
- ?? Container Classes
- ?? Forms Sets
- ?? Referencing Form Objects
- ?? The _SCREEN Object



This page intentionally left blank