Form Attributes **Sample Help**

Sample Description: Forms

Points of Interest

- Form Coordinate System
- Form Scale Modes
- Form Editing Environment
- Printing Text on a Form
- Creating a Custom Coordinate System

Control

- Form

For Help on Help, Press F1

Forms

Forms define a window on the screen and become a basis for building applications. You design an application by placing controls on forms. Most applications have more than one form. Most applications have several different forms that make up the foundation of the user interface.

Forms must be loaded into memory before they can be used. You can "Hide" forms when they are no longer needed. A loaded form must be explicitly "shown" to be made visible on the screen. This will automatically load the form into memory as well as execute any **Load** method.

The purpose of this sample is to provide some examples of the various properties that influence a form's behavior. You will learn about the coordinate system that supports form as well as how to activate and deactivate a form.

The sample form contains several controls that affect the behavior of the form. You may click and set some of these properties to see their effects on the form. You can resize the form by dragging one of the corners. This will update the Width and Height properties automatically.

Form Coordinate System

Every form has a coordinate system that contain an x- (horizontal) coordinate and a y- (vertical) coordinate. Such coordinates are often specified as an x,y pair and enclosed in parentheses. For example, (150, 65) identifies the pixel at x-coordinate 150 and y-coordinate 65. Notice that when two coordinate pairs have the same first value, the two pixels are on the same vertical line; when the second values are the same, the two pixels are on the same horizontal line.

By default, location (0,0) identifies the pixel at the upper left corner of the form. The x-coordinates increase moving horizontally to the right while the y-coordinates increase moving vertically down.

The default measurement units are *twips*. A twip is 1/1440 of an inch. The name stands for "twentieth of a point". So a graphical pixel point at (1440,0) is 1 inch to the right of the upper left corner. In the printing industry, a *point* is 1/72 of an inch. Common printing font sizes are 10-point and 12-point. Please keep in mind that twips really measure graphics distances when the object is printed. On two different monitors, the exact distance can vary somewhat between two points specified at the same pairs of coordinates.

Envelop provides several properties that allow you to customize the graphics scale and coordinate system for a form. You can modify the form's size with the *Height* and *Width* properties. Similarly, you can specify the form's on-screen location with the *Left* and *Top* properties. When you modify any of these properties, the form reflects the changes immediately.

Form Scale Modes

You can change the Scale Modes of a form from twips to something else. By adjusting the ScaleMode property, you can choose a different standard unit, such as inches or millimeters, or you can create a customized scale of your own. The possible Scale Mode values are listed below:

Value	Scale Units
0	Indicates a Custom Scale
1	Twip (default scale)
2	Point (1/72 of an inch, one point equals 20 twips)
3	Pixel (smallest unit of screen resolution)
4	Character (horizontal = 1/12 inch or 120 twips, vertical = 1/6 inch or 240 twips)
5	Inches
6	Millimeters
7	Centimeters

When you modify the values of a form's ScaleMode property, you don't change the size of the form or its location on the screen. All you do is change the scale that describes how many units Envelop creates to a given distance on-screen.

By changing the values of the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties in conjunction with ScaleMode, you can create a custom scale. The ScaleMode value of 0 means that a custom scale is in effect. Any time you modify the value of ScaleWidth, ScaleHeight, ScaleTop, or ScaleLeft (or use the Scale method), Envelop automatically sets the value of ScaleMode to 0.

When you change ScaleMode to a positive value, Envelop sets the values of both ScaleTop and ScaleLeft to 0. Envelop also modifies the values of ScaleWidth and ScaleHeight to reflect the new measuring units. In addition, Envelop updates the values of CurrentX and CurrentY to specify the last point referenced in the correct units

ScaleHeight and ScaleWidth

The Height and Width properties specify the size of a form. However, the total height and width of a form is not quite the same as the height and width of the form's drawing area. The **Height** property measures the total height of the form. This height includes the title bar and the widths of the horizontal borders. Similarly, the **Width** property measures the total width, which includes the widths of the vertical boundaries. The actual working area, however, lies in the form's interior. This working area lies inside all external borders.

To measure the available working area, Envelop uses the **ScaleHeight** and **ScaleWidth** properties. You can use ScaleWidth and ScaleHeight in your program code to accurately determine locations for printing text, placing controls, or drawing graphics. For example, the following instruction moves a button to the center of the form regardless of the size of the form.

```
Sub Button1_Click()
        Button1.Move ScaleWidth / 2 - Width / 2, ScaleHeight / 2 - Height / 2, Button1.Width, Button1.Height
End Sub
```

Regardless of the size of the form, Button1 will always move to the center of the form when it is clicked.

ScaleLeft and ScaleTop

Modifying ScaleWidth and ScaleHeight leaves the upper boundary at 0,0 and produces a customized scale. You can change the values of the boundary coordinates with the **ScaleLeft** and **ScaleTop** properties. For example, setting ScaleLeft to 500 and ScaleTop to 750, the coordinates of the upper left corner of the form become 500,750.

By using ScaleLeft and ScaleTop in conjunction with ScaleHeight and ScaleWidth, you can completely designate a customized coordinate system.

Form Editing Environment

Under Envelop's **ObjectEditor** object is a subclass known as **FormEditor**. The FormEditor object is used to modify the form editing environment. There are several properties used to modify the form editing environment. Many of these properties are toggled on/off by some of the icons in the Tools palette or in Envelop's main Tool Bar.

Editing

The Editing property can be set to True or False. When set to True, the current form (the one that has focus) can be edited. When a form is in edit mode, clicking on one of its controls will not execute the program code associated with that control, rather, it will update the Method and Property Browsers to highlight and display that particular control.

When the Editing property is set to False, the form is considered to be in "run" mode. At this point, you can click various controls on the form and see the results. Run mode is how an end user would see the application, while edit mode is how a developer would create or modify existing program code.

An icon in Envelop's main tool bar is used to toggle the run mode on/off. Regardless of whether the run mode is on or off, you can still debug program code with the Methods Editor and review and set various properties in the Property Editor. This is one of the benefits of an application such as Envelop that is designed to execute without requiring the application to be compiled first.

GridOn

The GridOn property is used to toggle the form's grid on and off. When the form's grid is toggled on, controls will "snap" to the nearest grid intersection. When toggled off, controls may be dragged freely around the form and when the Mouse button is released, will remain at their exact location.

The form's grid can only be active when the form is in Edit mode, or its Editing property set to True. On the sample form, a checkbox entitled "Grid On/Off" may be clicked to see the form's grid. However, in order to do this, you must first temporarily toggle the Editing property to True. Once the grid is displayed, a message box is posted. When you click the OK button on the message dialog, the form will automatically be placed back into "run" mode.

GridX, GridY

The GridX, GridY properties are the size of the spacing in the form's grid. The spacing is relative to values set in the GridX and GridY properties. In the sample form, scrollbars appear beside the GridX and GridY values. Clicking the scrollbars will change the values of the GridX and GridY properties as well as display the current setting on the form. Use the scrollbars to change the grid properties, then click the "Grid On/Off" checkbox to see the grid.

HitMode

The HitMode property is used to determine how controls are selected on the form. A control may be selected by clicking on it. In addition, the Mouse can drag a bounding box around several controls to select them. There are two types of hit modes:

HitMode Type	Description
1-RgnContains	Indicates the bounding box must entirely contain a control before it will be selected.
2-RgnTouches	Indicates the bounding box only needs to touch a portion of a control before the control will be selected.

An icon in the Tools palette is used to toggle the HitMode property.

NumSelected

The NumSelected property indicates the number of controls that are currently selected. If you were to drag a bounding box around 17 controls and select them all, the number 17 would appear in this property.

ShowOrder

The ShowOrder property is used to toggle the form's tab order display on and off. When the form's tab order is toggled on, a small white box will appear beside each control. When toggled off, the tab order boxes will disappear.

The form's tab order can only be displayed when the form is in Edit mode, or its Editing property set to True. On the sample form, a checkbox entitled "Show Order" may be clicked to see the forms tabbing sequence. However, to be able to do this, you must temporarily toggle the Editing property to True. Once the tab order is displayed, a message box is posted. When you click the OK button on the message dialog, the form will automatically be placed back into "run" mode and the tab order will disappear.

Printing Text on a Form

Printing text on a form is a matter of first specifying the X,Y coordinates you wish to print text, then issuing the **Print** statement. This can be done in the following code example:

```
CurrentX = ScaleWidth / 2
CurrentY = ScaleHeight / 2
Form1.Print "Envelop"
```

As you can see, the **CurrentX** and **CurrentY** properties are used to locate printing or graphical activities on the form. For each form, Envelop uses the CurrentX and CurrentY properties to specify the current location of the "last point referenced."

Printing in an application may also be redirected to another window, known as the **Debug** window. This may be done through the following instruction: Debug.Print "Envelop".

Creating a Custom Coordinate System

You can create a customized scale within a form. To do this, you modify the values of the ScaleWidth and ScaleHeight properties. The modification can be done when the form is being created or through program code at run-time. For example, if you execute the following instruction, the horizontal coordinates of Form1's working area range from 0 to 500.

```
Form1.ScaleWidth = 500
```

The instruction does not affect the size of the form on-screen, nor the location on-screen. What happens is that the right-hand border of the form area now has an x-coordinate of 500 while the left-hand border remains at 0.

If you make the ScaleWidth positive, the x-coordinate values increase as you move toward the right. When ScaleWidth is negative, the x-coordinate values decrease from left to right. In both cases, the x-coordinate of the left boundary remains at 0. As a result, when ScaleWidth is negative, the x-values go from 0 at the left boundary to negative values as you move toward the right.

Similarly, you can modify the value of ScaleHeight to customize the vertical scale. When you set ScaleHeight to a positive value, the y-coordinate values increase as you move down the form. If you make ScaleHeight negative, the coordinates decrease as you move downward. The y-coordinate of the upper boundary remains at 0.

A customized scale can be quite convenient. For example, a standard VGA screen has a resolution of 640 pixels by 480 pixels. Within the form, you can model this resolution with the following instruction:

```
Form1.ScaleMode = 0
Form1.ScaleWidth = 640
Form1.ScaleHeight = 480
```

Now, regardless of the form's physical size on-screen, references to the form's x,y-coordinate mimic drawing on a full VGA screen.

The values of Height and Width are always measured in twips. No matter what scale or measuring unit you specify with ScaleHeight and ScaleWidth, the measuring scale for the Height and Width properties remains unaffected.

You can also create a customized coordinate system with a single instruction using the **Scale** method. Using the Scale method is an efficient technique any time you want to change the coordinate system through your program code. For example, the following instruction creates a custom coordinate system:

```
Form1.Scale (0, 100) - (100, 0)
```

When you execute the Scale method, Envelop automatically updates the values of ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop. If you execute the Scale method without specifying any coordinate values, Envelop restores the coordinate system to the default scale.