
Scale and Resize Your Access Forms

Access 2000 Developer's Handbook, Volume I (Sybex)
Ken Getz, Paul Litwin, and Mike Gilbert
ISBN: 0-7821-2370-8

This is version 1.0.0 of this document.

Because so many Access developers need this functionality, we've decided to release a protected version of the form resizing code that's available as part of Access 2000 Developer's Handbook, Volume I. Included in this package you'll find a number of files:

- This document, in PDF format.
- ADHResize2K.MDE (the version for Access 2000)
- ADHResize97.MDE (the version for Access 97)
- ADHResizeTest2K.MDB (a sample database for Access 2000)
- ADHResizeTest97.MDB (a sample database for Access 97)

You are free to use and distribute this MDE file with any Access 2000 or Access 97 application you create. You may not, however, take the existence of these MDE files as giving you the right to distribute freely the original source code, should you happen to have purchased a copy of its original source (Access 2000 Developer's Handbook, Volume I). The copyright issues involved in distributing the code from the book still apply—see the book's Introduction for more information.

This document explains, in some detail, how to make use of the resizing code in the MDE files. Note that the Access 97 version of this MDE uses the same technology as the Access 2000 version—in other words, if you're currently using code from the Access 97 Developer's

Handbook, you'll need to call this code differently, and it will behave slightly differently.

WARNING: This document and the associated code, are provided completely without warranty, support, or other claims. That is, although you cannot harm your application directly using this tool, it's possible that it may crash, or otherwise cause your application to stop, just as with any other piece of code in Windows. Under no circumstances will the authors, book publisher (Sybex) or anyone else associated with this tool be liable for any damage, actual or construed, that occurs because of, or appears to occur because of, this tool. Here is the standard Sybex disclaimer, which applies to this code just as it applied to the code for the entire book:

Disclaimer

SYBEX makes no warranty or representation, either expressed or implied, with respect to this media or its contents, its quality, performance, merchantability, or fitness for a particular purpose. In no event will SYBEX, its distributors, or dealers be liable to you or any other party for direct, indirect, special, incidental, consequential, or other damages arising out of the use of or inability to use the media or its contents even if advised of the possibility of such damage. (We have to include this, just in case some crazy person decides that the loss of their data is somehow our fault. It's not.)

Automatically Resizing Forms

If all you are about is how to use the resizing code, you can jump directly to the section titled "Using FormResize", later in this article. If you want to know why and how it all works. continue reading.

When you set up Windows to run on your computer, you must choose a screen driver for use with your hardware. Your choice of screen driver allows your monitor to display a specific screen resolution, usually 640×480 (standard VGA), 800×600 (Super VGA), 1024×768 (XGA, Super VGA, or 8514/a), or 1280×1024. These numbers refer to the number of picture elements (*pixels*) in the horizontal and vertical directions.

If you create forms that look fine on your screen running at 1024×768, those same forms may be too large to be displayed by a user who's working at 640×480. Similarly, if you create forms at 640×480, someone working at 1280×1024 will see them as very small forms. (A full-screen form created at 640×480 takes up about a quarter of the screen at 1280×1024—although this is not necessarily something your users will want to change. Many people who use large displays and high-resolution adapters appreciate the fact that they can see not only a full-screen form, but other Access objects at the same time.)

One unattractive solution to this problem is to create multiple versions of your forms, one for each screen resolution you wish to support. This, of course, requires maintaining each of those forms individually. The following sections deal directly with the resolution issue. We present a class module you can use to scale your forms as they load, allowing them to look reasonable at almost any screen resolution. In addition, once you've solved the original problem, it's not difficult to extend this so the code allows users to resize a form and all its controls at runtime.

The sample form, frmScaleTest, demonstrates the technique of resizing a form to fit your screen resolution at load time. It also allows you to resize all the controls on the form as you resize the form. To try this out, load the form and try it out. Figure 1 shows a “mocked-up” image, containing the same form displayed at two different sizes at once. If you want to see now what's involved in making this happen, open frmScaleTest in Design view and check out the code in its module. Most of what you find there is comments—it takes very little effort on your part to get forms to scale. Basically, you must instantiate an object, set a property or two, and it works.

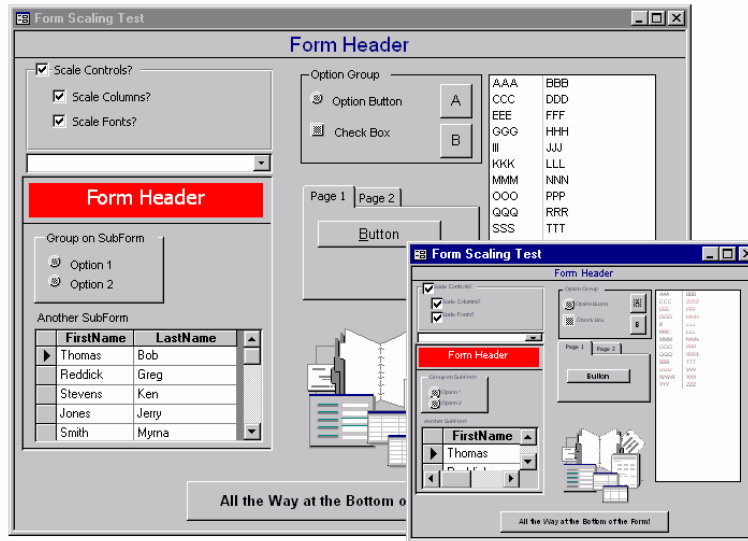


Figure 1: Two instances of the same form, one at full size and the other scaled to a smaller size

Understanding Screen Resolutions

Before you can understand the solution to the screen resolution issue, you must understand the problem. Figure 2 shows a scale image of the four standard Windows screen resolutions, superimposed. As you can see, a form that appears full screen at 640×480 will take up only a small portion of a 1280×1024 screen, and a full-screen form at 1024×768 will be too large for a screen at 800×600.

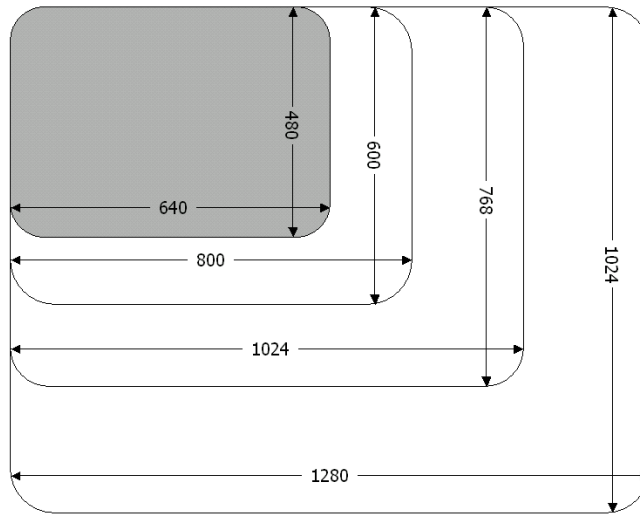


Figure 2: All four standard screen resolutions, superimposed.

The difference in the number of pixels is only one of two issues you need to consider in scaling forms. You must also think about the size of the pixels—the number of pixels per logical inch of screen space. Each screen driver individually controls how large each pixel is in relation to what Windows thinks an “inch” is. Windows provides API calls to gather all this information, which we’ll need later in this section. For now, the information of concern is the number of twips per pixel. (A *twip* is equivalent to 1/1440 inch.) Practical experience shows that screens at 640×480 use 15 twips per pixel, and all other VGA screen resolutions use 12 twips per pixel (although this isn’t a requirement, nor is it always true). This means that at low-resolution VGA, 100 pixels take up 1500 twips (a little more than one logical inch), while at higher resolutions, 100 pixels take up 1200 twips (a little less than one logical inch). Therefore, to correctly scale your forms for different resolutions, you need to take both ratios into account. You need to compare, for both the screen on which the form was prepared and the screen on which it will be displayed, the pixels used and the twips-per-pixel value. The ratios of these values control how you scale the form.

The resizing tool includes the code necessary to scale your forms at load time and to allow resizing by users at run time. This code makes

extensive use of Windows API calls to retrieve information about the current display and the sizes of forms.

Scaling Forms as They Load

To solve the problem of displaying forms so that they take up the same proportion of the screen real estate on different screen resolutions, it would seem that all you need do is calculate the ratio of the original screen dimensions to the current screen dimensions and scale the form accordingly. Unfortunately, the calculation is further complicated by the twips-per-pixel issue. Because different screen resolutions use a different number of twips for each pixel, you must also take this into account when calculating the new size for the form. The x-axis sizing ratio, when moving from 640×480 to 1024×768, is not just 1024/640. You must multiply that value by the ratio of the twips-per-pixel values. (Think of it this way: as far as Windows is concerned, pixels are “bigger” at 640×480. At higher resolutions, a pixel takes up fewer twips.) Figure 3 shows a single form, 400×120 pixels, created in 640×480 resolution, as it would display on a screen in 1024×768 resolution. The first example shows it unscaled, and the second example shows it scaled.

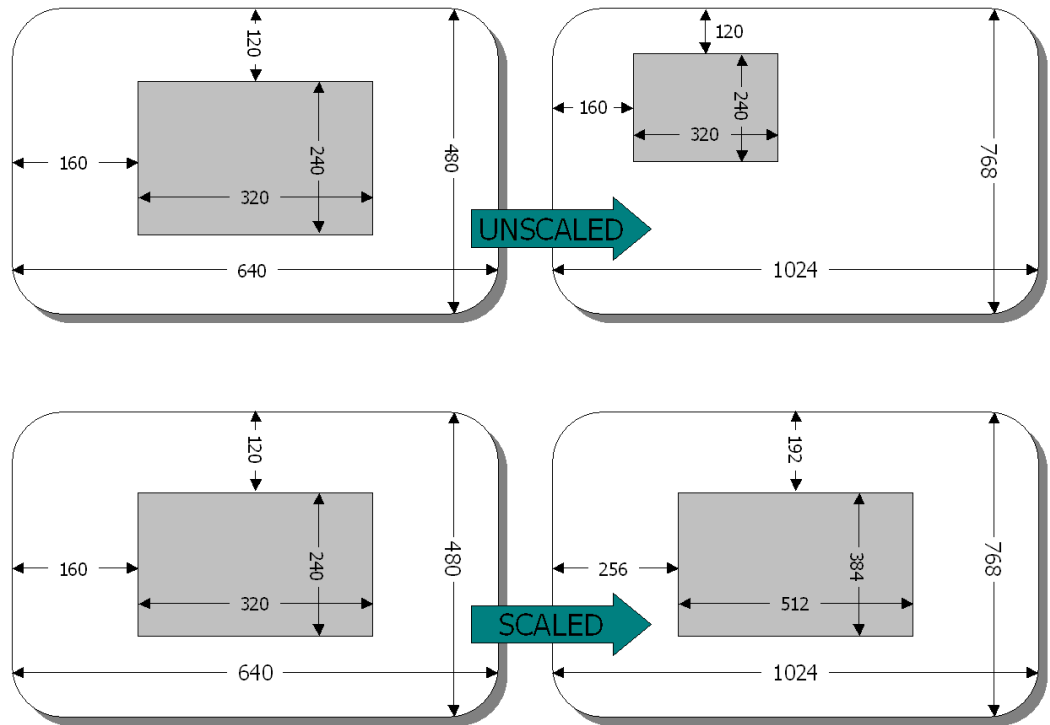


Figure 3: Scaling a form causes it to appear approximately the same on screens with different resolutions.

Necessary Information

To correctly scale your form, the code needs to know the screen resolution at which you created your form (so it can calculate the ratios between the screen widths and heights). It also needs to know the logical dots-per-inch values for the vertical and horizontal dimensions of the screen where you created the form. You may know offhand the screen resolution you use on your development machine, but you're unlikely to know the logical dots-per-inch values. Therefore, we've provided the frmScreenInfo form, shown in Figure 4. This form has one purpose in life: it provides information about your current screen settings so you can correctly call the SetDesignCoords method described below.

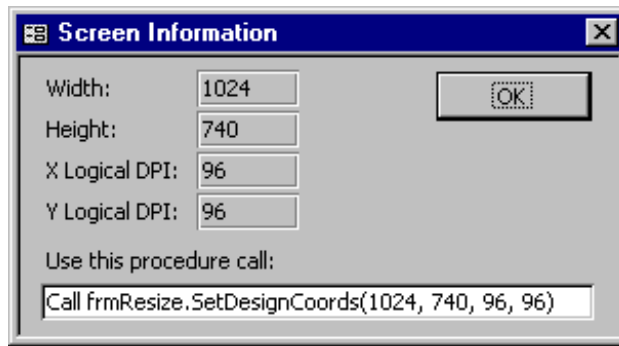



Figure 4: Use frmScreenInfo to calculate necessary screen coordinate information.

The code in this form's class module calculates the current size of the screen, taking into account the area chewed up by taskbars docked to the edges of your screen. It also calculates the logical dots-per-inch values and formats the method call as you'll need for your form. Once you've run this form, cut the value in the text box to the clipboard and paste it into your form's Open event procedure code. (The frmScreenInfo form does assume that you've named your FormResize variable as frmResize. If you change that name, you'll need to modify the method call, as well.)

☞ The information form, frmScreenInfo, takes taskbars into account when it calculates the screen resolution. If your users don't display their taskbars, you might want to set all taskbars on your system to be hidden when not in use. (Usually, this is the AutoHide property for the application.) That way, at worst, your form will scale too small (if you don't show taskbars but users do).


Using FormResize

From your application's perspective, the FormResize class manages all the scaling of your form and its controls. Once you've set up a connection between your form and its "shadow" FormResize object, the code in the class module handles all the work for you.

 If you're using Access 97, replace all the file names with the appropriate Access 97 version (ADHResize97.MDE, in this case).

Before you can use the code in this MDE, you'll need to set a reference to the appropriate MDE file. To do that, follow these steps:

1. Open a module in Design view.
2. Use the Tools|References menu from within VBA to open the References dialog box.
3. Click the Browse button.
4. From the Files of type: combo box, select MDE files. (If this isn't available in your version, type *.MDE in the File Name text box.)
5. Search for the location where you placed ADHResize2K.MDE, and select it once you find it. (Click the Open button on the Add Reference dialog box, once you've found the file.)
6. Back in the References dialog box, make sure that the ADHResize2K file is selected, then click OK.

 Be careful about references. Just like any other Access reference, you'll have problems if you move the MDE file, or when you deploy the whole application. Make sure you understand how Access handles external references (documented and discussed in "Access 2000 Developer's Handbook, Volume I" and in online help) before deploying an application that uses an external component.

In order to hook up your form, you'll need to add a little code to the form's module. You must, at least, follow these steps (if you're using Access 97, replace ADHResize2K with ADHResize97, wherever you see the reference):

1. To the form module's Declarations area, add a variable declaration for the FormResize object. Most likely, you'll want it to be Private:

```
Private frmResize As ADHResize2K.FormResize
```

2. In the form's Open event procedure, you must instantiate the FormResize object, and tell the object which form to shadow:

```
Private Sub Form_Open()  
    Set frmResize = ADHResize2K.CreateFormResize  
    Set frmResize.Form = Me  
End Sub
```

Why call CreateFormResize, instead of using "As New"? Because the FormResize class exists in a referenced library, it's difficult (and the technique is undocumented) to create a public, creatable class. To work around this, ADHResize2K.MDE includes a class whose only purpose in life is to return a reference to one of the FormResize objects.

If you go no further than that, your form:

- Will *not* perform scaling at load time. That is, it will not redimension itself to fit a changed screen resolution. (See the ScaleForm property in Table 1.)
- Will resize all its controls in reaction to the user resizing the form's border. (See the ScaleControls property in Table 1.)
- Will scale fonts for text box, combo box, list box, label, command button, toggle button, and tab controls so that they'll appear correctly in resized controls. (See the ScaleFonts property in Table 1.)
- Will scale columns in multicolumned list and combo box controls. (See the ScaleColumns property in Table 1.)

If you want to allow the form to scale its size, position, and contents at load time, based on its original design-time dimensions, you need to take one extra step: you must call the SetDesignCoords method of the FormResize object, indicating the screen size and dots/inch ratios on your design machine. As discussed in the "Necessary Information" section above, you can use the sample form frmScreenInfo to retrieve information about your screen, as you create your form. With this step added, your Open event procedure might look like this:

```
Private Sub Form_Open()  
    Set frmResize = ADHResize2K.CreateFormResize()  
    Set frmResize.Form = Me  
    Call frmResize.SetDesignCoords(1280, 1024, 96, 96)  
End Sub
```

☞ One of the benefits of using a class module to contain all this code (besides the fact that it allows the code to react to events of your form) is that you can attach the FormResize class to as many forms as you like, open them all at once, and have each form scale and resize individually. Although VBA loads multiple copies of the data—that is, each instance of the FormResize class in memory has its own property values—it only loads a single copy of the code. Using this technique is efficient and makes coding easier.

Working with FormResize Properties

Once you've set up the code, as in the previous section, you can programmatically control the behavior of the FormResize object. By setting various properties of the object, you can retrieve information about the form, make the form resize in just the way you want, or control the state of the form. Table 1 lists all the properties of the FormResize object, and Table 2 lists the public methods of the object. Some of these properties are somewhat subtle. (Properties and methods marked in **bold** are new for this version—that is, they don't exist in the code provided in the Access 2000 Developer's Handbook, Volume I.) For example:

- The Controls property returns a collection of ControlResize objects. Each of these objects has a Control property, which returns a reference to the control being shadowed. Although you can work through the FormSize's Controls collection to get to a particular control on your form, you're better off using the form's Controls collection instead. You'll generally only touch three properties of a ControlResize object: ScaleIt, FloatIt, and SizeIt, described in the next section.
- The four Scale... properties (ScaleColumns, ScaleControls, ScaleFonts, ScaleForm) allow you to control various behaviors of the form's resizing. All these properties are Boolean values except the ScaleControls property, which has three possible values: scYes, scNo, and scAtLoad. Choosing scAtLoad tells the FormResize class to scale controls at load time only, and then never again. This option is useful if you want to make sure your

form displays correctly at load time, resizing the form and its controls based on the screen resolution, but from then on, you want to keep the controls the size they were at load time.

- The `MaxWidth`, `MaxHeight`, `MinWidth`, and `MinHeight` properties allow you to set minimum and maximum sizes for the form. Using these properties (all measured in twips), you can specify the range of sizes for your form. In addition, if you specify values less than 1 (that is, fractional values) the code interprets these as indicating fractions of the available space. That is, if you specify a `MinWidth` property of 0.25, the form will never be allowed to be narrower than one-fourth of the client area (or the screen, if it's a pop-up form). (If you need to convert from twips to pixels, or back, you can use the `TwipsPerPixelX` and `TwipsPerPixelY` properties.)

The following sample code sets up many of the form `FormResize` properties (some of these are redundant—they're setting values to match the defaults):

```
Private Sub Form_Open(Cancel As Integer)
    Set frmResize = ADHResize2K.CreateFormResize()
    Set frmResize.Form = Me
    Call frmResize.SetDesignCoords(991, 721, 96, 96)
    frmResize.ScaleFonts = True
    frmResize.ScaleForm = True
    frmResize.ScaleColumns = True
    frmResize.ScaleControls = scYes
    frmResize.MinWidth = 0.1
    frmResize.MaxWidth = 0.9
    frmResize.MinHeight = 0.1
    frmResize.MaxHeight = 0.9
End Sub
```

Table 1: FormResize Properties

PROPERTY	TYPE	DESCRIPTION
CenterOnOpen	Boolean	Center the form when it's opened. Works around conflicts that occur when you turn on the <code>AutoCenter</code> property—you really can't use that property here. (New in this version—not in the book.)
Controls	Collection	(Read-only) Collection of <code>ControlResize</code>

		objects contained within the FormResize object. Generally, you won't need to work with this collection, but it's available for your convenience. Under no circumstances should you add or delete anything from this collection in your code.
Form	Form	Sets or retrieves the reference to the real form that the FormResize class is associated with. No other properties or methods will work correctly until you've set this property.
HeightInTwips	Long	(Read-only) Returns the current height, in twips, of the associated form.
IsMaximized	Boolean	Sets or retrieves the maximized state of the associated form. Set the property to True in order to programmatically maximize the form.
IsMinimized	Boolean	Sets or retrieves the minimized state of the associated form. Set the property to True in order to programmatically minimize the form.
MaxHeight	Single	Sets or retrieves the maximum height of the associated form. If greater than 1, specifies the maximum height in twips. If less than or equal to 1, specifies the percentage of the available space to fill (the MDI client for normal forms, the screen for pop-up forms).
MaxWidth	Single	Sets or retrieves the maximum width of the associated form. If greater than 1, specifies the maximum width in twips. If less than or equal to 1, specifies the percentage of the available space to fill (the MDI client for normal forms, the screen for pop-up forms)
MinHeight	Single	Sets or retrieves the minimum height of the associated form. If greater than 1,

		specifies the minimum height in twips. If less than or equal to 1, specifies the percentage of the available space to fill (the MDI client for normal forms, the screen for pop-up forms).
MinWidth	Single	Sets or retrieves the minimum width of the associated form. If greater than 1, specifies the minimum width in twips. If less than or equal to 1, specifies the percentage of the available space to fill (the MDI client for normal forms, the screen for pop-up forms).
ScaleColumns	Boolean	Set to False to disable scaling of column widths within combo and list boxes (the default value is True).
ScaleControls	ScaleControlsWhen	Set to scNo (0) to disable scaling of controls on the form. Set to scAtLoad (1) to cause controls to be scaled when you first load the form, and then never again. Set to scYes (-1) (the default) to always scale controls in relation to the shape of the form.
ScaleForm	Boolean	Set to False to disable scaling of the form to match its original screen size. This doesn't disable scaling of controls when you resize the form at runtime, only the automatic scaling of the form to match its original shape.
ScaleFonts	Boolean	Set to False to disable scaling of fonts on the screen (the default value is True).
TwipsPerPixelX	Long	(Read-only) Returns the ratio between twips and pixels in the horizontal direction for the current screen driver. Can be used to convert from twips to pixels, when necessary.
TwipsPerPixelY	Long	(Read-only) Returns the ratio between twips and pixels in the vertical direction for the current screen driver. Can be

		used to convert from twips to pixels, when necessary.
WidthInTwips	Long	(Read-only) Returns the current width, in twips, of the associated form.
Version	String	Returns a version identification string. (New in this version. Not in the book.)

Table 2: FormResize Methods

METHOD	PARAMETERS	DESCRIPTION
Center		Center the form on the screen. (New—not in the book.)
RescaleForm		Forces a recalculation of control and font sizes. If you change a property manually (ScaleFonts or ScaleColumns, for example) you'll want to call this method to force a recalc of the form's display.
SetDesignCoords	Width, Height, DPIX, DPIY	Optionally, call this method to indicate the original, design-time screen coordinates. If you don't call this method, FormResize will assume that the design-time coordinates match the runtime coordinates, and no automatic scaling at load-time will occur. Use frmScreenInfo to gather the necessary information for this method call at design time. (Width and Height represent coordinates of the screen, and DPIX and DPIY represent the dots/inch in the horizontal and vertical directions.).

Managing Features on a Control-by-Control Basis

In some cases, you won't want to scale each and every control on a form. You may want to scale some controls but leave others the same size they were when you created them, no matter how the end-user mangles your form's shape. In order to "turn off" scaling, you have several options:

- You can set the FormResize object's ScaleControls property to scNo at design time. No controls will ever scale.
- You can set the FormResize object's ScaleControls property to scNo at any time while the form is running. Doing this will temporarily turn off scaling of the controls on the form, as the user resizes the form. (Change the property back to scYes when you want scaling to start again.)
- You can leave the ScaleControls property alone but modify properties of each individual control for which you'd like to disable scaling.

This section provides the details for taking the third option and adds some new functionality along the way, as well.

Properties of the ControlResize Object


In order to do its work, the FormResize object keeps track of information about each control on the form with which it's associated. To manage the information, it uses the ControlResize class to create a ControlResize object corresponding to each control. This object keeps track of items such as the control's name, its coordinates, and its parent (a FormResize object). Most of the object's public methods and properties are public only so that they can be used from the parent object, but it does include some properties that you'll find useful. Table 3 describes each of the properties of the ControlResize object that you're likely to use.

Table 3: Useful Properties of the ControlResize Object

PROPERTY	DATA TYPE	DESCRIPTION
FloatIt	ControlFloat: cfRight, cfBottom, cfBoth, cfNone*	As you resize the form, the code can float the control towards the right, bottom, or both. (The size won't change.) The distance between the upper-left corner of the control and the specified edge of the form will remain constant.
ScaleIt	ControlScale: csYes, csNo, csDefault	As you resize the form, the code can both float and size the control based on the size of the form (this is the standard rescaling behavior). You can control this on a control-by-control basis, using this property. If you specify csDefault, the control will resize based on the settings you've made for the entire form. Otherwise, you can disable or enable scaling for each particular control.
SizeIt	ControlSize: czRight, czBottom, czBoth, czNone	As you resize the form, the code can size the control towards the right, bottom, or both (the upper-left corner of the control won't move). The distance between the upper-left corner of the control and the specified edge of the form will remain constant.
*Default values marked in bold. All values are members of Enums, in ControlResize's class module.		

As you can see from Table 3, you have more flexibility than simply controlling which controls scale. If you want to control the scalability of individual controls, you can set the ScaleIt property of any control, like this:


```
' mfr is a FormResize object, previously  
' instantiated.  
mfr.Controls("cmdCancel").ScaleIt = csNo
```

 If you want to programmatically control scaling, sizing, or floating, you must use the Controls collection of the FormResize object, not the form itself. Controls in the form's Controls collection don't have ScaleIt, SizeIt, and FloatIt properties—only objects based on the ControlResize class have those properties. Members of the FormResize's Controls collection are based on the ControlResize class, so you'll need to use that Controls collection instead.

Using code as in the previous fragment, you'll be able to manage, on a control-by-control basis, whether any specific control should scale to match the size of the form.

You can also declare a variable of the correct type, and then set it to be the control you want to work with, like this:

```
Dim x As ADHResize2K.ControlResize  
Set x = mfr.Controls("cmdCancel")  
x.FloatIt = True
```

 The ControlResize ScaleIt property always overrides the FormResize object's ScaleControls property. Even if you've set the ScaleControls property to False, setting an individual ControlResize object's ScaleIt property to scYes will cause that control to be scaled.

In addition to the ScaleIt property, the ControlResize class also provides two other useful properties that aren't really linked to scaling at all. Using the FloatIt and SizeIt properties, you can control the positioning and sizing of controls in relation to the lower right-hand corner of the form.

For example, Figure 5 shows two instances of the same sample form (frmFloatAndSize, from the sample project). As the form grew larger, the controls didn't scale—they actually moved, or resized, to fit the larger space. What's the difference between scaling, floating, and sizing?

- When scaling, the controls' left and top coordinates change, and normally, fonts change size, as well. (You can control this using the FormResize object's ScaleFonts property.) You can control

when scaling occurs: always, never, or only when the form first loads.

- When sizing, the controls' left and top coordinates stay fixed, but the width and height change to maintain a constant offset from the bottom and right edge of the form. You can control the direction of the sizing towards the bottom, towards the right, neither, or both.
- When floating, the controls' top and left coordinates change to maintain a constant offset from the bottom and right edge of the form, but the width and height of the controls remain fixed. You can control the direction of the floating towards the bottom, towards the right, neither, or both.

In Figure 5, all the controls include one or more of these settings. The form's Load event includes the following code, which sets up all the values (mfr is the module-level variable that refers to the FormResize object):

```

mfr.Controls("txtMain").SizeIt = czBoth
mfr.Controls("cmdTest").FloatIt = cfBoth
mfr.Controls("cmdOK").FloatIt = cfRight
mfr.Controls("cmdCancel").FloatIt = cfRight
With mfr.Controls("lblStatus")
    .FloatIt = cfBottom
    .SizeIt = czRight
End With

```

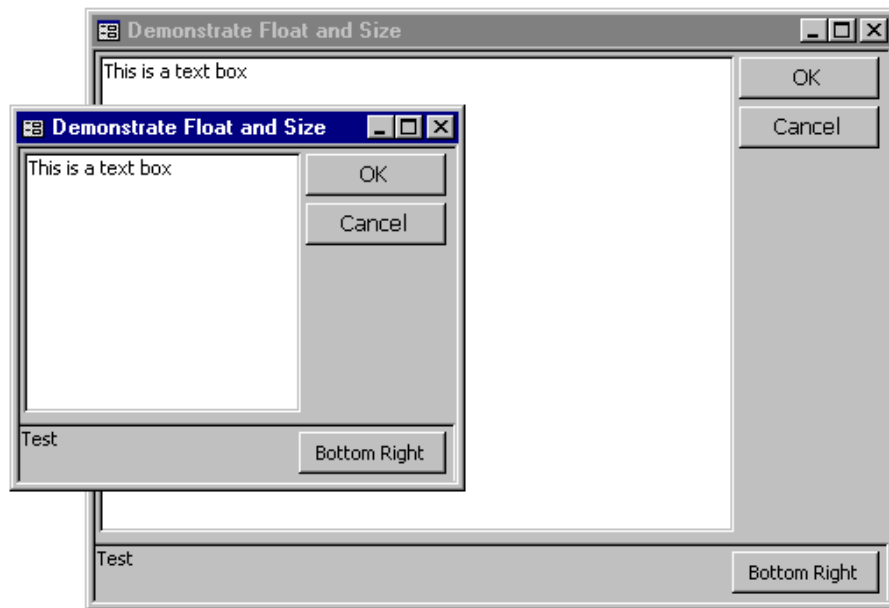


Figure 5: Using the FloatIt and SizeIt properties, you can cause controls to float and size in relation to the lower-right corner of the form.

Looking carefully at the property settings, you can work through the details:

- The form sets the ScaleControls property for the associated FormResize object to be scNo, so the controls don't scale to fit the sized form. The ScaleForm property is set to False, as well, so the form doesn't attempt to scale to fit the current screen resolution at load time.
- The text box (txtMain) sizes in both directions as you resize the form. Its upper-left corner doesn't move, but its width and height change.

- The Bottom Right command button (cmdTest) floats in both directions. Its upper-left corner moves to the right and down, as you resize the form. Its width and height never change.
- The two command buttons (cmdOK and cmdCancel) float only to the right. This means that their Left properties change but not their Top properties. Because they're floating, their width and height never change.
- The sunken label (lblStatus) at the bottom of the form floats with the bottom of the form (that is, its Top property changes, but not its Left property). In addition, it floats with the right edge of the form (that is, its Width property changes, but not its Height).

As you might guess, using these control-level properties gives you immense flexibility in the way you create forms. Even if you use none of the scaling features provided by the FormResize class, being able to float and size controls based on the changes made to the size of the form can make your life as a programmer simpler.

Using the Tag Property to Manage Scaling

Because you're most likely to want to set the FloatIt, SizeIt, and ScaleIt properties for your controls once and never modify them again, you may want a way to set these properties at design time. That way, you needn't write any code to set the properties at runtime. Unfortunately, controls don't normally have FloatIt, SizeIt, or ScaleIt properties. To work around this problem, we've set up the resizing code so that you can specify these properties as part of the Tag property of each control.

The class that manages the scaling of each individual control includes code that checks the Tag property of each control as it's initializing information about the form and keeps track of the values it finds there. Using the standard technique described in Chapter 7 of *Access 2000 Developer's Handbook, Volume I*, (the same chapter number in *Access 97 Developer's Handbook*), the code looks for portions of the Tag property containing text like this:

```
ScaleIt=Yes;FloatIt=Yes
```

That is, the code looks for a property name, an equal sign, and a value for the property. If you set up the Tag property for each affected control this

way, you needn't write any code in the form's Load event procedure, as shown in the previous section. The sample form, frmFloatAndSizeTag, uses this technique to produce the same results as the sample shown in the previous section. For a complete list of tag names and possible values, see Table 4.

Table 4: Possible Tag Names and Values for Sizing, Scaling, and Floating Individual Controls

TAG NAME	POSSIBLE TAG VALUES	DESCRIPTION
FloatIt	Right, Bottom, Both, None*	As you resize the form, the code can float the control towards the right, bottom, or both. (The size won't change.) The distance between the upper-left corner of the control and the specified edge of the form will remain constant.
SizeIt	Right, Bottom, Both, None	As you resize the form, the code can size the control towards the right, bottom, or both (the upper-left corner of the control won't move). The distance between the upper-left corner of the control and the specified edge of the form will remain constant.
ScaleIt	Yes, True, On, No, False, Off, Default	As you resize the form, the code can both float and size the control based on the size of the form (this is the standard rescaling behavior). You can control this on a control-by-control basis using this tag value. If you specify Default, the control will resize based on the settings you've made for the entire form. Otherwise, you can

		disable or enable scaling for each particular control.
*Default values marked in bold.		

Steps to Successful Scaling

If you're upgrading to the current version of this technology from a previous version of this book, you'll find that the resizing algorithm used in this book works much better than it did in previous versions. We no longer see the troublesome round-off errors when resizing fonts that plagued previous versions of the code.

Even though the code works better in this version, there are still some rules you must follow to make it possible for this code to work:

- Use TrueType fonts for each control you will scale. This code will scale only the fonts in text box, combo box, list box, label, command button, toggle button, and tab controls. Unfortunately, the default font used in all controls is not scalable. You must either modify your form defaults or select all the controls and change the font once you're finished designing. On the other hand, beware of using fonts that won't be available on your users' machines. All copies of Windows 95 and NT ship with Arial and Times Roman fonts; choosing one of these for your buttons and labels and list, combo, and text boxes guarantees a certain level of success. Of course, all the Office applications use the Tahoma font, and you may wish to use this font in order to "blend in" with the rest of Office.
- Do not design forms at 1280×1024 and expect them to look good at 640×480. By the time forms get scaled that far down, they're very hard to read. Using 800×600 or 1024×768 for development should provide forms that look reasonable at all resolutions.
- The current implementation of this code cannot resize subforms shown as datasheets. We tried vainly to accomplish this—there's simply too much information we need that isn't available about the row and column sizes to make this possible. You should be aware that the contents of datasheets will not scale, although their physical size will.

- Do not attempt to mix the AutoCenter property with the ScaleForm property of a FormResize object set to True. The AutoCenter property will attempt to center the form before it's scaled and will cause Access to place the form somewhere you don't expect it to be.
- Make labels and text boxes a bit wider than you think you actually need. Windows doesn't always provide the exact font size the code requests, so you're better off erring on the generous side when you size your controls.

Scaling Your Own Forms

To include this functionality in your own applications, follow these steps:

1. Copy ADHResize2K.mde (or ADHResize97.mde, if you're using Access 97) to a convenient location.
2. In your database, use the Tools|References menu to set a reference to the appropriate MDE file. You'll need to use the Browse button to find the file—it won't appear on the list automatically.
2. Ensure that all the fonts on your form are scalable. (Use TrueType fonts if possible, since they're all scalable.)
3. In the Declarations area of the form module for each form you'd like to scale, declare an object variable to refer to the FormResize object that will mirror your form (the actual name doesn't matter, of course):


```
Private frmResize As ADHResize2K.FormResize
```

Or

```
Private frmResize As ADHResize97.FormResize
```

3. In the form's Open event procedure, set your object variable to be the result returned from calling the CreateFormResize method. Then, set the object's Form property to be the current form, like this:


```
Set frmResize = ADHResize2K.CreateFormResize
Set frmResize.Form = Me
```

 If you've been using the version of this code that's provided in the Access 2000 book, you have been using the New operator to instantiate a new instance of the FormResize class. This is difficult to do when creating an object from an Access library, so this version includes the CreateFormResize function to do that work for you.

4. If you intend to scale the form so that it appears proportional to the screen, as it did when you designed it, you must also call the SetDesignCoords method. Pass to the method the x- and y-resolutions of the screen for which it was designed, along with the logical dots-per-inch values for the horizontal and vertical dimensions of your screen. (Use frmScreenInfo to generate this line of code.)

```
Private Sub Form_Open(Cancel As Integer)
    ' Instantiate the class to handle all resizing.
    Set frmResize = ADHResize2K.CreateFormResize

    ' Tell the new object what form you want
    ' it to work with.
    Set frmResize.Form = Me

    ' Tell the object the size of the screen
    ' on which you designed the form, and the
    ' dots/inch in that screen resolution.
    ' Replace these four integers with your own
    ' values. Use frmScreenInfo to calculate
    ' these for you.
    ' If you don't call this method at all,
    ' the code will display the form as you
    ' originally designed it, with no scaling
    ' at load time.
    Call frmResize.SetDesignCoords(1024, 740, 96, 96)
End Sub
```

5. If you'd like, set any/all of the optional FormResize properties, such as ScaleControls, ScaleForm, ScaleFonts, or ScaleColumns. (See Table 1.)
6. If you want to control floating, scaling, or sizing of individual controls, either set their Tag properties appropriately, or write

code in the form's Load event to handle these individual properties. (See the sample forms, frmFloatAndSize, and frmFloatAndSizeTag).

- Code in the FormResize class sets the OnResize and OnLoad event properties of your form to be "[Event Procedure]". If you have other values in those properties already, they will be overwritten at runtime. That is, if you're calling a macro or a function from those properties, the macro or function won't be called. Instead, the code in the FormResize class will run. If you're already calling an event procedure from the Resize or Load event (that is, you've set the OnLoad and OnResize properties to be "[Event Procedure]", your code will run first, and then the code in the FormResize class will run.