VSVBX 5.0

QuickStart Tutorial

To learn how to use help, press F1

vsElastic

Smart containers that resize themselves and their child controls, automatically create labels and 3-D frames for its child controls, and can also be used as progress indicators and labels.

vsIndexTab

Allows you to group controls by subject, using the familiar notebook metaphor that has become a Windows standard.

vsAwk

Parsing engine named and patterned after the popular Unix utility, plus a powerful expression evaluator.

Elastic QuickStart

The vsElastic control is extremely versatile. You can use it as a one-stop replacement for a number of Visual Basic's built-in controls and other custom controls, or you can use it to provide your applications with functionality that would be difficult or impossible to achieve with Visual Basic code alone.

Basic vsElastic Features

You can use the vsElastic as an enhanced replacement for the following Visual Basic controls: Picture Boxes, Image controls, Labels, Command Buttons, and Frames. To find out how, look up the following properties in the reference section of this manual:

To replace:	***Look up these properties:***
Image Controls	Picture, PicturePos
Label Controls	Caption, CaptionPos, AccessKey, WordWrap, TagLabel
Command Buttons	Style, BevelOuter, BevelOuterDir, CornerColor
Frame Controls	Style
Progress Indicators	FloodColor, FloodDirection, FloodPercent

Across all operation modes, the vsElastic provides a variety of 3D effects for its own caption and borders, as well as 3D borders for all controls placed inside the vsElastic (standard VB controls and 3rd party controls included).

To find out more about vsElastic's 3D effects, look up the following properties in the reference section of this manual: BevelOuter, BevelOuterWidth, BevelInner, BevelInnerWidth, BorderWidth, and BevelChildren.

Dynamic Resizing

The vsElastic has two properties that govern all its automatic resizing features, both at design time and at run time.

The **Align** property determines how the vsElastic controls resizes **itself** when its parent is resized.

The **AutoSizeChildren** property determines how the vsElastic resizes its **children** (the controls it contains) when it is resized.

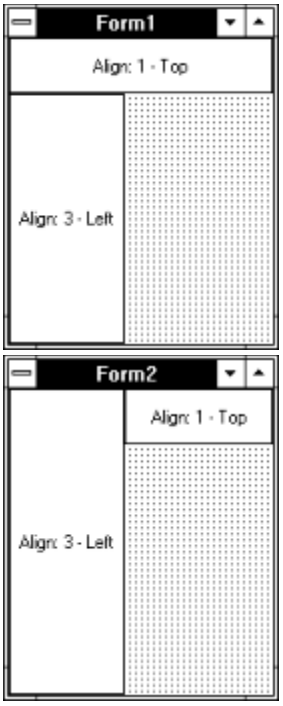
After you learn how these properties work, you will be able to design resolution-independent forms that rearrange themselves when the user resizes them. And the vsElastic can also help you at design time, by automatically aligning, sizing, and distributing groups of controls.

The Align Property

The vsElastic's Align property is similar to the standard Align property, like the one the Picture Box control has, but with additional settings and capabilities. Like the standard Align property, the vsElastic's Align lets you align the control to the top, bottom,left, and right of a form. In addition, the vsElastic property lets you align controls to fill the entire form when it is resized.

The other difference is that the standard Align property lets you align to forms only, while the vsElastic's Align property lets you align to any container control, including other vsElastics.

There is only one thing to remember when using the vsElastic Align property: if several vsElastics within the same container have their Align property set, the ones further behind get aligned first, and subtract their area from the container. The picture below illustrates this:



Both forms shown above have two vsElastics, one aligned to the top and one to the left. The difference is that on Form1, the vsElastic aligned to the top was created first. It aligned itself to the top and reduced the free area on the form. The second vsElastic aligned itself to the left of the remaining area.

On Form2, the vsElastic aligned to the left was created first. It aligned itself to the left and reduced the free area on the form. The second vsElastic aligned itself to the top of the remaining area.

This does not mean that you have to create the vsElastics in perfect order to use the **Align** property. Visual Basics Edit menu has options that allow you to send controls to the back or bring them to the top, changing the order in which they are aligned.

For example, if you selected the vsElastic aligned to the left on Form1, then clicked on the Edit| Send to Back Visual Basic menu option, it would become identical to Form2. You can even change the order of the controls at run time, using the standard ZOrder method.

Here is an important conorder of this discussion: **vsElastics with the Align property set to 5 (Fill Container) should always be brought to the front of the form**, so they get aligned last. Otherwise, theyll eat up all the space available on the container and there will be no room left for the other vsElastics.

The AutoSizeChildren Property

While the **Align** property governs resizing the vsElastic within its container, the AutoSizeChildren property govern resizing of the controls **inside** the vsElastic.

Before we start describing what the **AutoSizeChildren** property does, let us describe a couple of things it **does not** do:

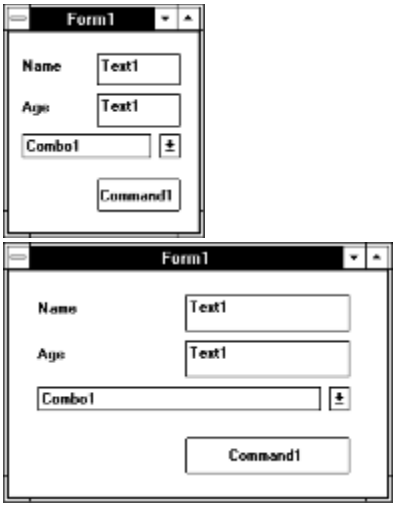
1) The vsElastic does not automatically resize fonts. There are two main reasons for this: (1) resizing fonts every time a form is resized would be very time-consuming, and (2) there is no general way to decide how to resize fonts when one dimension of the control grows and the other shrinks at the same time. If you want your application to use fonts proportional to screen size, you can easily do it by adding some code to the Form_Load event.

2) Certain controls cannot be freely resized. Combo Boxes, for example, have a fixed height, and the standard List Box can only be resized to certain heights that depend on the font being used. The vsElastic will align these controls within their limitations.

Ok, now lets see the things that AutoSizeChildren can do for you. We will cover the AutoSizeChildren settings one by one, starting with the simplest and most useful ones:

Proportional Sizing (AutoSizeChildren = 7)

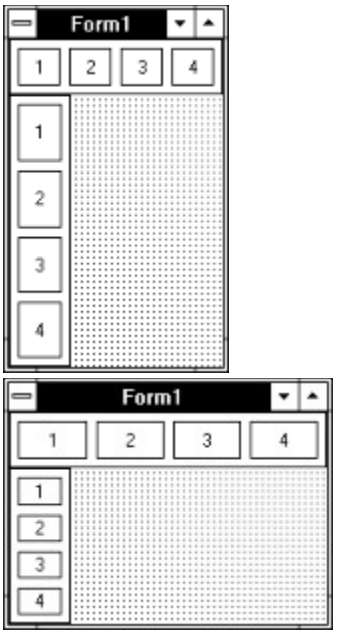
This is the easiest setting to explain: when the vsElastic get resized, all its children are resized proportionally. The picture below shows an example:



The form shown above has a single vsElastic control with the **Align** property set to *5 Fill Container* and **AutoSizeChildren** set to *7 Proportional*. All other controls are placed on the vsElastic. When the form is resized, all controls are also resized and retain their proportions.
Note that while **AutoSizeChildren** is set to proportional, the controls on the vsElastic cannot be moved or resized. If you try to move them, the vsElastic will put them back where they were. If you need to move controls around, set **AutoSizeChildren** to *0 None*, do all the editing you want, then set it back to *7 Proportional.*

Even Sizing (AutoSizeChildren = 1, 3)

Use these settings when you want to distribute controls evenly within the vsElastic. The picture below shows how these settings works when you resize a form:



The vsElastic aligned to the top of the form has its **AutoSizeChildren** property set to *1 - Even Horizontal,* and the vsElastic aligned to the left of the form has its **AutoSizeChildren** property set to *3 - Even Vertical*. Notice how they resize the controls inside them as the form gets resized.
Changing the order of the children with the vsElastic is easy: just drag controls to their new position with the mouse. The vsElastic will automatically sort and align them for you. Try it, the resizing works at design time too.
If you want, you can also change the order of the children at run time also. All you have to do is move the control with code just like you would with the mouse. For example, to move button 4 to the left of button 3 on the form shown above, you would use the following code:

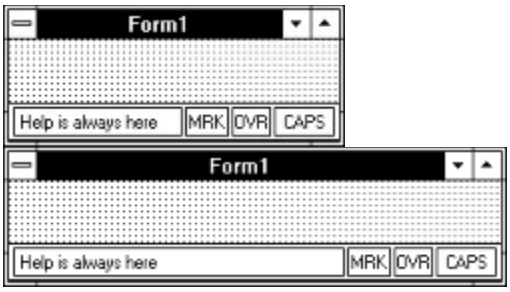
```
Button4.Left = (Button3.Left - Button3.Left) / 2
```

The above code would move button 4 to a position between buttons 2 and 3. The vsElastic would then take care of realigning the controls to the proper position.

Uneven Sizing (AutoSizeChildren = 2, 4)

Sometimes you don't want all the children controls to be the same size, or to remain proportional. A typical example is a status bar, where you may reserve a few fixed-size spots to show keyboard status and use the remaining space to show help messages and prompts.

The picture below shows a form with a status bar created with the vsElastic, before and after resizing:

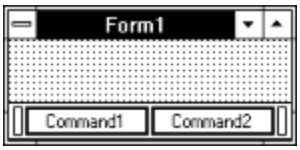


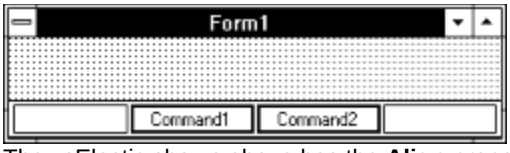
The vsElastic shown above has the **Align** property set to *2 Bottom* and the **AutoSizeChildren** property set to *2 Uneven Horizontal.* It also has a number of child controls used to display status information. When the form is resized, only the leftmost child get resized.
How does the vsElastic decide which child to resize? Simple: it is always the front control. So all you have to do to create unevenly spaced vsElastics is create all the children, size them appropriately, then decide which one you would like the vsElastic to resize for you, and bring it to the front using VBs Edit|Bring to Front menu option. Then set the **AutoSizeChildren** property to one of the uneven settings and youre done.
That is all there is to uneven spacing: when the vsElastic is resized, the front child gets stretched to fill the gap left by the others.

Elastics Only Sizing (AutoSizeChildren = 5, 6)

The *Elastics Only* settings tell the vsElastic to stretch only child controls which are also vsElastics. This is a combination of the settings described above. If all child controls are vsElastics, these setting are equivalent to *Even*. If only the top child control is a vsElastic, these settings are equivalent to *Uneven.*

When would you need the *Elastic Only* settings? For example, if you want to center clusters of controls on a form, as shown below.





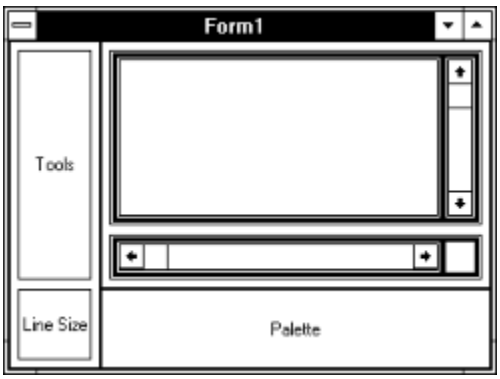
The vsElastic shown above has the **Align** property set to *2 Bottom* and the **AutoSizeChildren** property set to *5 Elastics Only Horizontal*. It also has four children: two vsElastics and two command buttons between them. When the form is resized, the vsElastics are resized and push the command buttons to the center of the container.

Using Nested vsElastics

Now that you know how the vsElastic works, you are ready to learn one more trick we haven't discussed yet: *nested* vsElastics.

It takes a little practice, but once you get the hang of it you will be able to create all sorts of forms without worrying about what happens to your layout if the user decides to resize the form.

For example, the picture below show the anatomy of a form similar to the Windows Paintbrush application built with five vsElastics:

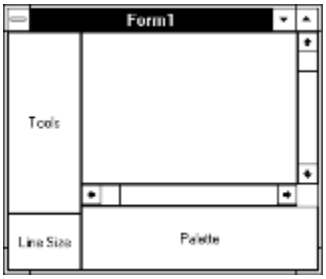


Can you tell where the five vsElastics are? Heres a list describing their positions, contents, and properties:

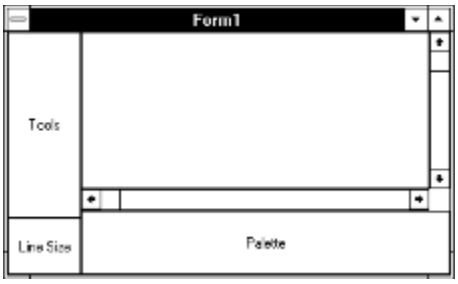
1. Aligned to the left of the form, with the Tool Bar and Line Width controls on it. Its AutoSizeChildren property is set to *Uneven Vertical*.
2. Aligned to the bottom of the form, with the Color Palette on it. Its AutoSizeChildren property is set to *Even Horizontal*.
3. Filling the rest of the form, with two black vsElastics on it. Its AutoSizeChildren property is set to *Uneven Vertical*.
4. A black vsElastic with a Picture Box and a vertical Scroll Bar on it. Its AutoSizeChildren property is set to *Uneven Horizontal*.
5. A black vsElastic with a horizontal Scroll Bar and a little Label Control on it, also set to *Uneven Horizontal* child sizing. The little label has the same width as the vertical Scroll Bar above it.

The final step is to set the BorderWidth and ChildSpacing properties of the vsElastics to zero, so they'll do their job without appearing on the form. After doing this, it becomes hard to select the vsElastics with the mouse. Heres a hint: select a control you can grab with the mouse, then use the tab key until you get the vsElastic you want.

This is what the final form looks like:



When the user resizes it, look what happens:



All controls resize automatically. Without vsElastics, you would have to write a lot of very boring code to do this, and resizing would be much slower.

IndexTab QuickStart

If you use Windows, chances are you have seen index tabs at work in several applications and know how to use them. The following sections describe how the vsIndexTab control works and how you can use it to implement index tabs on your applications.

How vsIndexTab Works

Before we get into our tutorial, let's quickly review how vsIndexTab works. This will make it easier for you to understand what is going on while you follow the steps indicated in the tutorial. If you are impatient and would like to start creating tabs right away, go ahead and skip this section.

The vsIndexTab acts like a stack of pages. At any time, you see and operate only on the top page. When you click on a tab, vsIndexTab automatically brings the corresponding page to the top of the stack.

To make this scheme work, the vsIndexTab uses two key concepts: tabs and pages.

Tabs Tabs are created with the Caption property. The Caption contains the text that goes in each tab, separated by pipe characters ("|"). For example, if you assign the string "Tab 1|Tab 2|Tab 3" to a vsIndexTab caption, you will have three tabs.

Pages Each tab needs its own page, which you must create yourself. Each page is a simply a container control (i.e., a control that allows you to drop other controls inside it.) The best container control for use with vsIndexTab is the vsElastic, because of its resizing capabilities. We will go over the process of adding pages in the tutorial.

vsIndexTab Tutorial

With vsIndexTab, it only takes a few minutes (and no code whatsoever) to create fully functional tab-based interfaces. This chapter takes you step by step through the development of a simple project.

Before you start following the tutorial, you need to start Visual Basic and load the VSVBX custom controls. For help on how to load custom controls, see the Visual Basic documentation.

Step 1 Create a new vsIndexTab control

Create a new vsIndexTab control on your form just like you would create any other control: click the tool for the vsIndexTab control on VB's Toolbox, then move the pointer onto the form. The pointer will become a cross hair. Place the cross hair near the top left corner of the form. Hold the left mouse button, drag the cross hair to a point near the bottom right corner of the form, then release the button.

The vsIndexTab appears on the form, as shown in the picture below:

Step 2 Create the tabs

To create the tabs, you need to set the vsIndexTab's Caption property. Select the vsIndexTab control you just created by clicking on it, then press F4 to bring up the property window. Highlight the Caption property and type the following:

Visual &Basic|Visual &C++|&Ole Controls

You have just created three tabs, complete with underlined accelerator keys. The text for each tab is delimited by pipe characters (vertical bars). Each tab has its own accelerator key, defined by the ampersands ("&").

The vsIndexTab should now look like this:

Step 3 Create a page for each tab

Click the tool for the vsElastic control on VB's Toolbox, then move the pointer onto the vsIndexTab, where it says "Place Container Control Here". Hold the left mouse button, drag the cross hair to a point near the bottom of the vsIndexTab, then release the button.

The vsElastic will automatically fill the tab area.

If this were a real application, now would be the time to design the first page. To speed up our tutorial, though, just place a single command button on the middle of the page.

The vsIndexTab should now look like this:

To create the next page, activate the second tab by double-clicking it with the right mouse button. If you are using software that interferes with the mouse, or if you are a keyboard-oriented person, you can activate the second tab by pressing F4 to bring up the property window and changing the vsIndexTab CurrTab property to 1.
The second tab will be activated, and since there's no second page yet, you will see the "Place Container Control Here" message again. Repeat the steps outlined above to create a new vsElastic for the second page. Place a new command button on it.

Repeat the above steps once again to create the third page.

The vsIndexTab should now look like this:

Note that you must create the pages in the proper order: first the page for tab 0, then the page for tab 1, and finally the page for tab 3. This is because whenever you see the "Place Container Control Here" message, the next container control created is assigned to the first tab that does not yet have a page, not to the current tab.

Step 4 Customize the Appearance of the tabs

The vsIndexTab offers extensive control over the appearance of the tabs. The easiest way to customize the tabs is to use the Template property to select a predefined style and then adjust it by setting individual properties.

Select the vsIndexTab control by clicking on it, then press F4 to bring up the property window (click on the tabs themselves, otherwise youll probably select the page instead of the vsIndexTab control). Highlight the Template property and set it to *4 - Berkeley*. This will automatically set the Position, Style, Font, and color properties.

The vsIndexTab should now look like this:

The skeleton of the vsIndexTab control is ready. If you want to test the control, run the application as it is and click on the tabs. The vsIndexTab will automatically switch pages whenever the current tab changes. You can also switch tabs using the accelerator keys or the arrow keys. Stop the application when you are done testing it.

Step 5 Design the Pages

The last step required is to design each page by placing on it the controls that your application requires. Of course, this step depends entirely on the nature of your application. Let's place a few controls on the first page just to illustrate the general procedure.

Switch to the first tab using the mouse or the Properties Window. Now you are ready to start working on the contents of the first page.

Start by eliminating the vsElastic's outer bevel, since the vsIndexTab itself provides a bevel. To do this, click on the vsElastic, then press F4 to bring up the property window. Highlight the BevelOuter property and set it to *0 - None*. Also, set the Form background color to light gray, so it blends with the vsIndexTab background.

Now place three text boxes on the right-hand side of the page. Align them to the top, center and bottom of the page, leaving room for the command button. Move the command button if you have to, and change its Caption property to "&Ok".

The vsIndexTab should now look like this:

Notice how the vsElastic automatically provides a bevel around the text boxes. Add some labels to identify the text boxes if you like.
When you are done with the first page, select the second tab as you did before, with the mouse or using the Properties Window. This will bring up the second page, and you can design it just like you did the first.
When you are done with all pages, select the tab that you would like to appear when your application starts, then save your project. This will ensure that when you start the application the current tab will be the one you want.
Congratulations! You have just finished our tutorial and created your first VideoSoft vsIndexTab!

Advanced vsIndexTab Topics

Editing the Tab Order

The above tutorial described how to create tabs from scratch. But what if you want to add tabs later, or to change the tab order for an existing vsIndexTab control? There is an easy way to do that also.

Say you want to change our little three tab tutorial application (described above) by adding a new tab right after the first one. The first thing you need to do is change the vsIndexTab Caption property to create the new tab. Set it to the following:

Visual &Basic|&Add-Ins|Visual &C++|&Ole Controls

Stretch the vsIndexTab horizontally so all tabs are visible. The vsIndexTab control supports automatic tab scrolling and multiple tab rows, but for now it will be easier to work if all tabs are visible.

The new tab, *Add-Ins*, has been created, but it does not yet have a page. If you start switching tabs now, you will notice there is a page missing. Because pages are assigned to tabs sequentially, the last tab lost its page.

To fix this, activate the last tab using the mouse or the Proeprties Window. Since there is a page missing, the "Place Container Control Here" message will appear again. Create the new page.

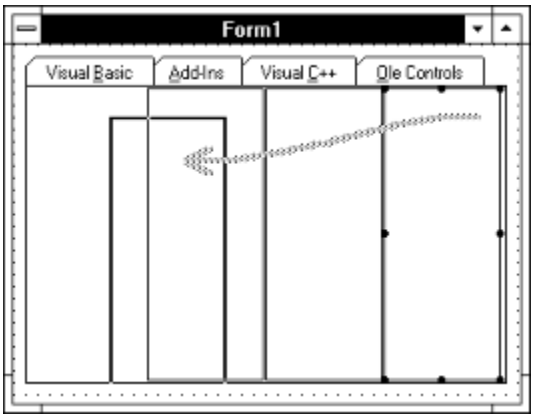
Now you have the right number of pages, but they are in the wrong order. Here comes the interesting part:

Set the CurrTab property to -1, either using the Properties Window or by clicking the tab that is already current. This will tile all pages within the vsIndexTab so they all become visible at once. Setting the CurrPage property to -1 means "show all pages at once."

If you have been following the tutorial, then you may think the first tab is missing. It isnt, but you cant see it because you set its BevelOuter property to *0 - None* in step 5 (remember?). You can still select it and work with it normally. All the buttons you created earlier are still there too, but they may be out of sight while the pages are tiled.

Now that you can see all pages in order, click on the last page (the one you just created) and drag it to its proper position (right after the first page). The vsIndexTab will tile all pages again so they remain organized.

The picture below shows how to do this:



You can keep dragging pages around until you get the page order you want. When you are happy with the order, activate the first tab again using the mouse or the Properties Window. Everything is back to normal, and the tab order is now correct.

Controls that appear on all pages

Sometimes it may be desirable to create controls that appear on all tabs (e.g. Ok and Cancel buttons). The easiest way to do this using the vsIndexTab is to create these controls outside the vsIndexTab and then just drag them over the tab area. Because the new controls do not belong to

any page, they will remain visible when you switch tabs.

Pages that repeat for several tabs

Sometimes you may want to use a single page and simply change the contents of the controls on it when the user switches tabs. To do this, define a single page and ignore the "Place Container Control Here" message (it only appears at design time).

At run time, the vsIndexTab will look for the page that corresponds to the current tab. If the current tab number is greater than the number of pages, it will show the last page.

Awk QuickStart

The Awk is the most unusual of the controls in VSVBX, because it is the least visual of the three. However, once you become used to its text parsing capabilities, you will find it is one of the most useful controls you've ever used.

The simplest way to use the Awk is to set its FileName property to the name of a file you want to work with, set the Action property to 1 (Scan) and supply a Scan event handler to collect the information you want. The Awk will read the file, one line at a time, parse the line into fields according to the FS property, and fire the Scan event for each line.

The following paragraphs show how you can use the Awk to parse files, grid clip strings, and how you can implement a user-friendly calculated text box.

Awk Use 1: Parsing Text Files

The following code counts the words in a text file, without any string or file statements:

```
VsAwk1.FileName = filename
VsAwk1.Action = 0 ' scan

' Gets called right after the file is opened
Sub VsAwk1_Begin ()
   NWords = 0
End Sub

' Gets called after each line is read
Sub VsAwk1_Scan ()
   NWords = NWords + VsAwk1.NF ' number of fields = number of words
End Sub

' Gets called right after the file is closed
Sub VsAwk1_End ()
   MsgBox filename + " has " + format(NWords) + " Words."
End Sub
```

Of course, you can access individual fields. Suppose, for example, that you wanted to add up all the values on the second field of a file imported from a spreadsheet program.. The code below does this:

```
VsAwk1.FileName = "spread.txt"
VsAwk1.Action = 1 ' scan

' Gets called right after the file is opened
Sub VsAwk1_Begin ()
   Total = 0
End Sub

' Gets called after each line is read
Sub VsAwk1_Scan ()
   Total = Total + Val(VsAwk1.F(2)) ' second field
End Sub

' Gets called right after the file is closed
Sub VsAwk1_End ()
   MsgBox "The total is " + format(Total) + " ."
End Sub
```

Awk Use 2: Parsing Grid Clip Strings

You can use the Awks parsing engine with any strings, not necessarily reading them from a file. To do this, just assign the string to the Awks L (Line) property, and read the fields one by one. This is especially useful when you store strings clipped from a grid control. Such strings contain fields separated by line breaks (chr(13)) and tabs (chr(9)). You can then use two Awks, one to split the rows and one to split the columns, like this:

```
VsAwk1.FS = chr(13)            ' to break the string into rows
VsAwk2.FS = chr(9)             ' to break each row into columns

VsAwk1 = clipstring
for i = 1 to VsAwk1.NF        ' VsAwk1.NF is the number of rows
   VsAwk2 = VsAwk1.F(i)        ' parse ith line
   for j = 1 to VsAwk2.NF     ' VsAwk2.NF is the number of columns
      A(i, j) = Val (VsAwk2.F(j))
   next
next
```

Awk Use 3: Creating a Calculated Text Box

You can use the Awks expression evaluator to create user-friendly data entry controls such as calculated text boxes. Your users will certainly appreciate being able to type 1234.23*0.085= in a Sales Tax text box instead of having to use a calculator.

The following code replaces expressions with their values whenever the user hits the equals (=) key:

```
Sub Text1_KeyPress (KeyAscii as Integer)
   If KeyAscii = Asc ("=") Then
      VsAwk1 = Text1
      v = VsAwk1.Val
      If VsAwk1.Error = 0 then
         Text1 = Format (v)
      Else
         Beep
      End If
   End If
Exit Sub
```