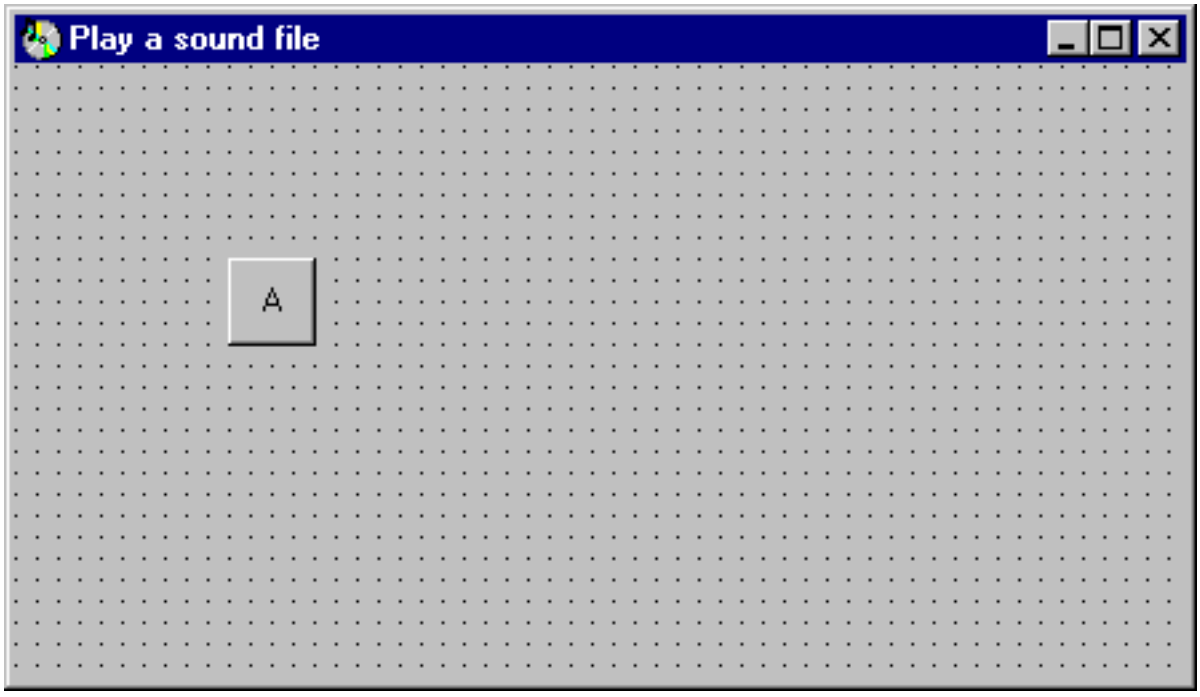
Play a Sound file in Visual Basic 6

My daughter recently received, as a gift, a wonderful learning game, and she's really learned a lot from it. However, once concern I have is that the first thing it does is ask her to use the mouse to spell her name from a displayed keyboard, but the keyboard is not in the typical "Q-W-E-R-T-Y" format. Because of that, I set out to write her a little Visual Basic program to permit her to spell her name, but to display the letters on the keyboard in the typical keyboard order.

With that exception, I wanted to duplicate the features of the other program, includes sounding out the letters as she uses the mouse to click on them. In addition, I also wanted to allow her to enter her name by typing the letters on the keyboard, something that the other program does not.

Design the Interface

My first step was to design the interface. I did this by creating a form, and placing a single Command Button on the form. I named it cmdLetter, and set its Caption Property to 'A' to represent the letter 'A' of the alphabet. As you can see, I also set the Form's Caption to 'Play a sound file' and specified a custom icon for the Icon property of the Form.



I need to display 25 other Command Buttons to represent the remaining letters of the alphabet, and this will be a perfect application for a Control Array. For those of you not familiar with Control Arrays, a Control Array is a collection of controls of the same type (i.e. Command Button,) with the same name (i.e. cmdLetter), but with unique values for the Index Property. The great thing about using Control Arrays is

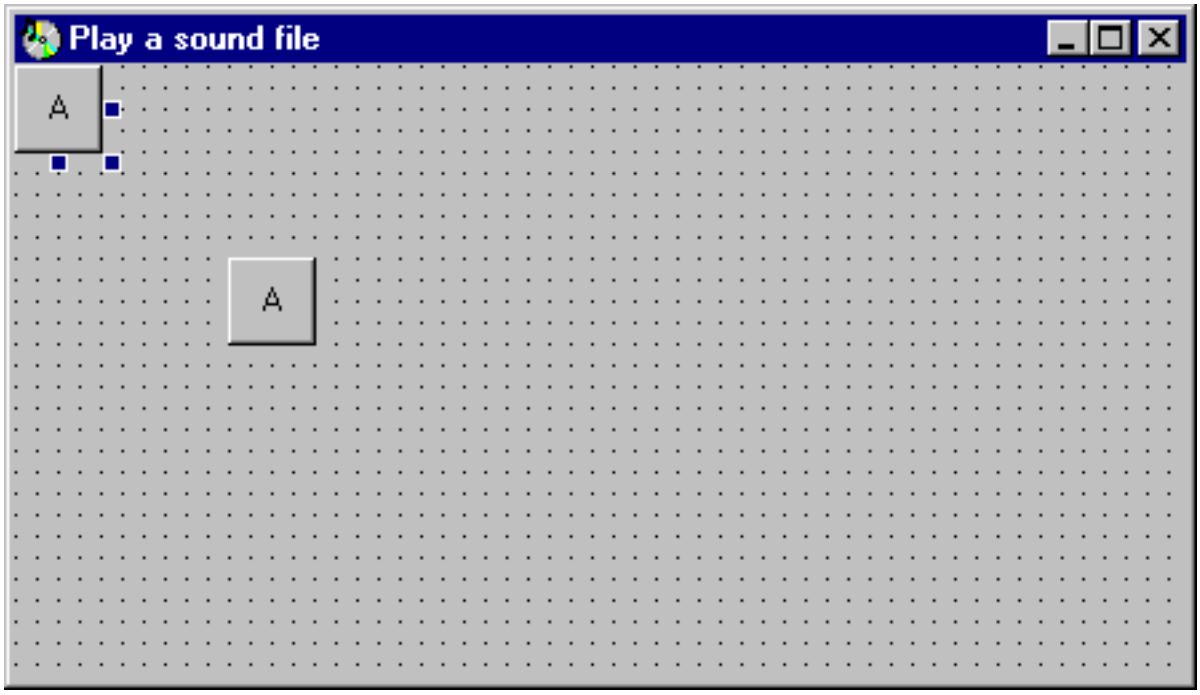
that each 'member' of the Control Array shares the same Event Procedure. What this means is that if I have 26 members of a Control Array called cmdLetter, each time I click on any one of the 26 Command Buttons, the same Click Event procedure will be executed.

Create a Control Array

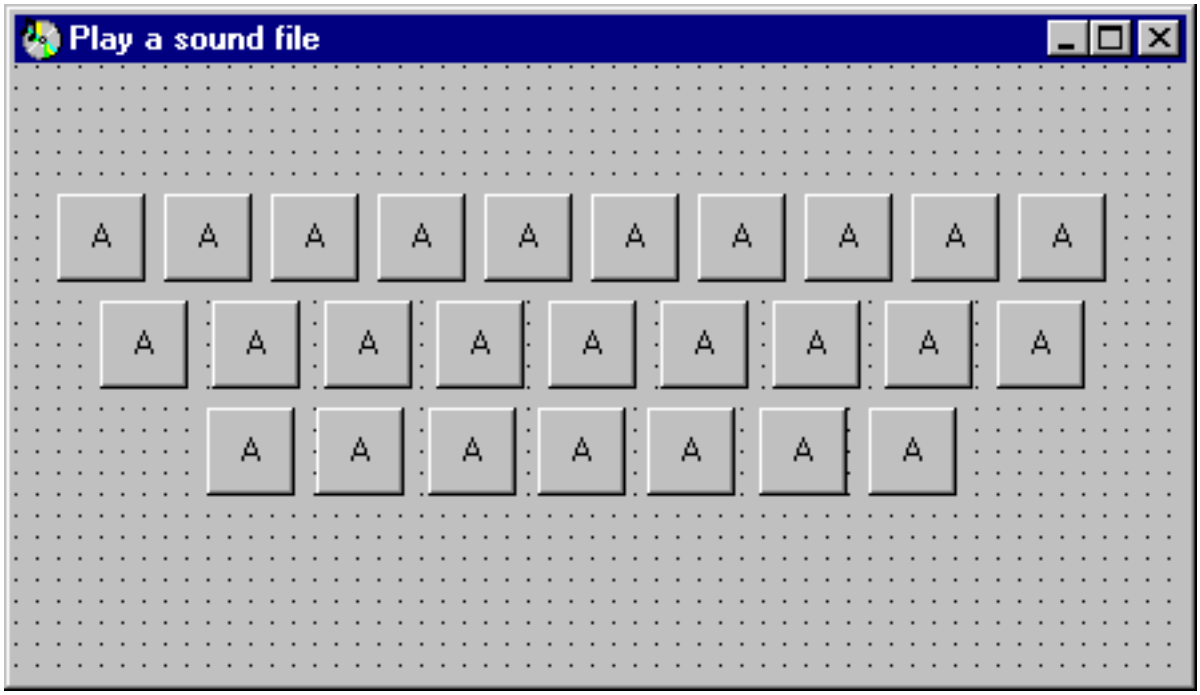
If what I want to do is take some action when the user (i.e. my daughter) clicks on the Command Button, a Control Array is ideal as I'll only need to write the code once--although I will need to be able to determine which of the 26 Command Buttons was clicked (this is easy, as you'll see in a little bit.)

Creating a Control Array is relatively easy---all I need to do is click on the Command Button, then select Edit-Copy from the Visual Basic Menu Bar, then click on the Form, and select Edit-Paste from the Visual Basic Menu Bar. When I do, this message is displayed…

By answering 'Yes' here, Visual Basic places a second Command Button in the upper left hand corner of the form, with the same name as the first (cmdLetter), but with an Index value of 1 (the first Command Button's Index value is set to 0)…

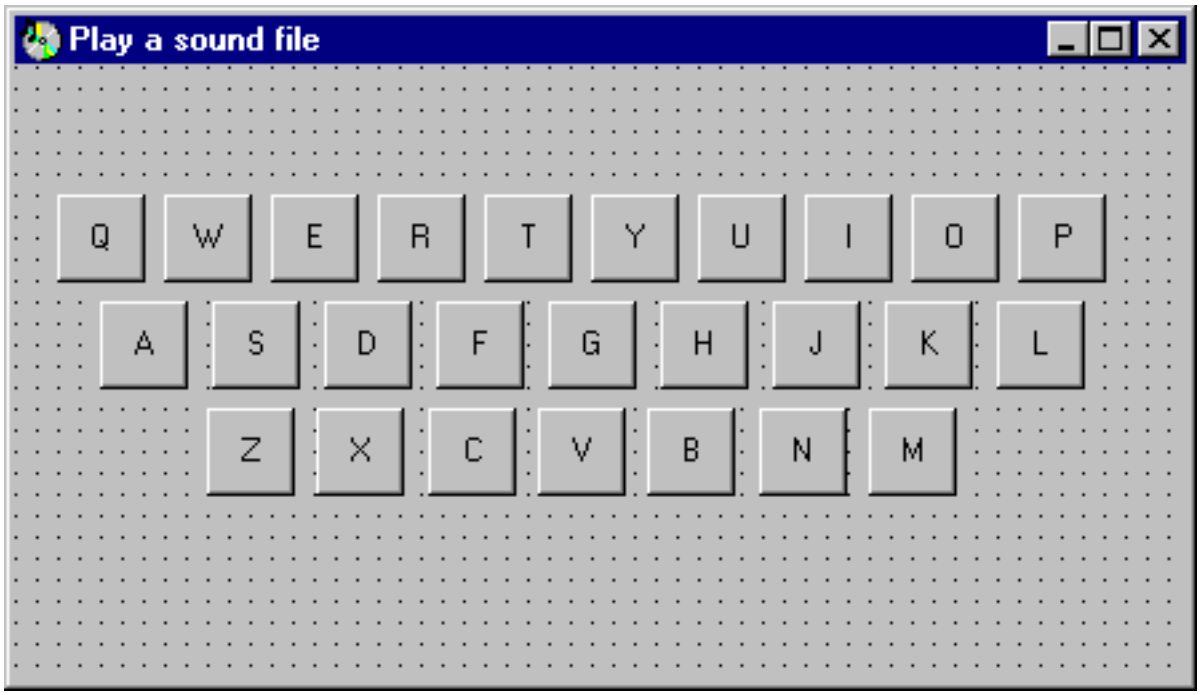


At this point, we need to place 24 'copies' of the first Command Button on the form, and after moving that second Command Button out of the way, can do that quickly by pressing Ctrl-V. We won't be asked about creating a Control Array this time as we already have one. After this, it's just a matter of pasting the 'copy' on the form, and arranging the 'letters' so they look like this…



Now that we have 26 Command Buttons on the form arranged to approximate the layout of the "Q-W-E-R-T-Y" keyboard, our next step is to change the Caption property of each one so that's they're in the correct order. At the same time, I want to ensure that the Index Property for the button captioned 'A' is 0, the Index Property for the button captioned 'B' is 1, and so on until we get to 'Z' which should

have a Caption property of 25. This will be very important later on.



That's better---the interface is now complete. Now it's time to write the code to 'play' the sound file.

Write the code to play the sound file

There are many ways to play a sound file in Visual Basic, but undoubtedly the easiest way is to use the Windows API to do so. Now you may have heard of the Windows API (in fact, I've written an article on using the Windows API here at Mightywords), and its name may sound foreboding, but using it is not really a big deal.

As it turns out, there is a Windows DLL by the name of "winmm.dll". A DLL is just a library of code that programs (typically written in C or C++) can access to perform Windows functions. If you think about it, what happens when Windows starts up. If you have a sound card in your PC, and if you have made the appropriate selections via the Control Panel, Windows will play sounds. How does it do that? Windows plays sounds by executing a function in the "winmm.dll" called PlaySound---and you can execute that same function from within your Visual Basic program. Here's how.

Declare the PlaySound() Function

Using the Windows API (which stands for Application Program Interface) simply requires that we tell Visual Basic the name of the function (or procedure) that we wish to 'call' and its location. We do that by using the 'Declare' statement in the

General Declarations Section of the Form. Here's the declaration for the PlaySound Function.

```
Option Explicit

Private Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" _
  (ByVal lpszName As String, ByVal hModule As Long, _
  ByVal dwFlags As Long) As Long
```

This code may look relatively complicate by Visual Basic standards, but it's not really bad at all. It's a long declaration, so I've broken it up into three lines by using the Visual Basic line continuation character, but as you can see, what we're doing here is declaring a Function called "PlaySound" and we're telling Visual Basic that it's located in the DLL (or Library--that's what 'Lib' means) called "winmm.dll". The word 'Alias' means that "PlaySound" is actually stored in the DLL as "PlaysoundA", but we can use the function in our Visual Basic program as "PlaySound".

The statement within the parentheses you are probably familiar with from your own Visual Basic code--these are arguments, or qualifiers for the Function, and PlaySound expects three of them.

LpszName is the name of the sound file to play, hModule is an argument whose value should be 0, and dwFlags tells PlaySound how to play the sound file---for our purposes, if we pass a value of 0 to PlaySound the sound file is played Synchronously--which means that our program waits for the sound file to complete playing. If we pass a value of 1 to Playsound, the sound file is played Asynchronously---which means that our program will not wait for the sound file to complete playing. This is something to bear in mind for sound files that play for a long duration.

Finally, since PlaySound is a function, it returns a value to the program that calls it, and its return value is a Long Integer. We'll need to bear this in mind when we write the code to 'call' PlaySound. For more information on the PlaySound function, you can check this Microsoft Web Site:

http://www.microsoft.com/officedev/articles/sound.htm

Call the PlaySound Function

Our next step is to write the code to call the PlaySound Function.

I should tell you ahead of time that I have a collection of 'wav' files for each letter of the alphabet--you can download my collection at

http://www.johnsmiley.com/downloads/lettersounds.zip

Each one of the letters is represented by a unique sound file---the letter 'A' is A.WAV, the letter 'B' is B.WAV, etc.

Bearing that in mind, here is the code I wrote to play the sound corresponding to the letter of the alphabet that my daughter clicked on…

```
Private Sub cmdLetter_Click(Index As Integer)

Dim Retval As Long

Select Case cmdLetter(Index).Caption
 Case "A"
  Retval = PlaySound("C:\VBFILES\SOUNDS\A.WAV", 0, 0)
 Case "B"
  Retval = PlaySound("C:\VBFILES\SOUNDS\B.WAV", 0, 0)
 Case "C"
  Retval = PlaySound("C:\VBFILES\SOUNDS\C.WAV", 0, 0)
 Case "D"
  Retval = PlaySound("C:\VBFILES\SOUNDS\D.WAV", 0, 0)
 Case "E"
  Retval = PlaySound("C:\VBFILES\SOUNDS\E.WAV", 0, 0)
 Case "F"
  Retval = PlaySound("C:\VBFILES\SOUNDS\F.WAV", 0, 0)
 Case "G"
  Retval = PlaySound("C:\VBFILES\SOUNDS\G.WAV", 0, 0)
 Case "H"
  Retval = PlaySound("C:\VBFILES\SOUNDS\H.WAV", 0, 0)
 Case "I"
  Retval = PlaySound("C:\VBFILES\SOUNDS\I.WAV", 0, 0)
 Case "J"
  Retval = PlaySound("C:\VBFILES\SOUNDS\J.WAV", 0, 0)
 Case "K"
  Retval = PlaySound("C:\VBFILES\SOUNDS\K.WAV", 0, 0)
 Case "L"
  Retval = PlaySound("C:\VBFILES\SOUNDS\L.WAV", 0, 0)
 Case "M"
  Retval = PlaySound("C:\VBFILES\SOUNDS\M.WAV", 0, 0)
 Case "N"
  Retval = PlaySound("C:\VBFILES\SOUNDS\N.WAV", 0, 0)
 Case "O"
  Retval = PlaySound("C:\VBFILES\SOUNDS\O.WAV", 0, 0)
 Case "P"
  Retval = PlaySound("C:\VBFILES\SOUNDS\P.WAV", 0, 0)
```

```
  Case "Q"
    Retval = PlaySound("C:\VBFILES\SOUNDS\Q.WAV", 0, 0)
  Case "R"
    Retval = PlaySound("C:\VBFILES\SOUNDS\R.WAV", 0, 0)
  Case "S"
    Retval = PlaySound("C:\VBFILES\SOUNDS\S.WAV", 0, 0)
  Case "T"
    Retval = PlaySound("C:\VBFILES\SOUNDS\T.WAV", 0, 0)
  Case "U"
    Retval = PlaySound("C:\VBFILES\SOUNDS\U.WAV", 0, 0)
  Case "V"
    Retval = PlaySound("C:\VBFILES\SOUNDS\V.WAV", 0, 0)
  Case "W"
    Retval = PlaySound("C:\VBFILES\SOUNDS\W.WAV", 0, 0)
  Case "X"
    Retval = PlaySound("C:\VBFILES\SOUNDS\X.WAV", 0, 0)
  Case "Y"
    Retval = PlaySound("C:\VBFILES\SOUNDS\Y.WAV", 0, 0)
  Case "Z"
    Retval = PlaySound("C:\VBFILES\SOUNDS\Z.WAV", 0, 0)
End Select

End Sub
```

Let's take a closer look at this code now. As I mentioned earlier, because each one of the 26 Command Buttons is a member of a Control Array, there is only one Click Event Procedure for which we need to write code. For those of you who haven't seen a Control Array before, take note of the Click Event Procedure header---it contains one extra piece of information that the header for an ordinary Command Button would not contain--and that's an Index parameter…

```
Private Sub cmdLetter_Click(Index As Integer)
```

Remember, each one of the Command Buttons has a unique Index value---and when any one of the Command Buttons is clicked, that value is 'passed' to the Click Event Procedure. We can use that information to determine which one of the 26 Command Buttons has been clicked, and we do that within a Select…Case statement…

```
Select Case cmdLetter(Index).Caption
```

This single line of code uses the Index parameter, which contains the Index property of the Command Button that the user (in this case my daughter) clicks. If she clicks on the letter 'Q', the Index value for that Command Button is passed to the Click Event Procedure. I happen to know that the Index Property of the letter Q

is 17, and I could have used that to determine which sound file to play--however, it's much easier to base the selection on the Caption property of the Command Button that has been clicked and that's what the Select Case statement line is telling Visual Basic to evaluate--the Caption property of the Command Button whose Index value equates to the Index property passed to this Event Procedure.

After that, individual Case Statements are used to determine which sound file to 'play' using this syntax

```
Case "A"
  Retval = PlaySound("C:\VBFILES\SOUNDS\A.WAV", 0, 0)
```

to 'call' the PlaySound function. Notice that since PlaySound is a function, we need to assign its return value somewhere--and we do that by assigning it to the declared Long variable Retval (we don't need to do anything with the return value, it's just informational, and if the function call is successful, should return a value of 0.)

We pass three arguments to PlaySound--the first argument is the name of the file to play (notice that we include its full path name), the second argument is ordinarily a zero, and the third argument is 0 indicating that we want this sound file to be played 'synchronously'.

If we now run this program, provided we have a sound card installed and our sound files in the correct folder on our hard drive, we should hear the letter 'A' when the user clicks on the letter 'A'.

What's Next?

We can streamline this code just a bit by taking advantage of the fact that the caption of the Command Button being clicked is the same as the name of the sound file to be played. Take a look at this code…

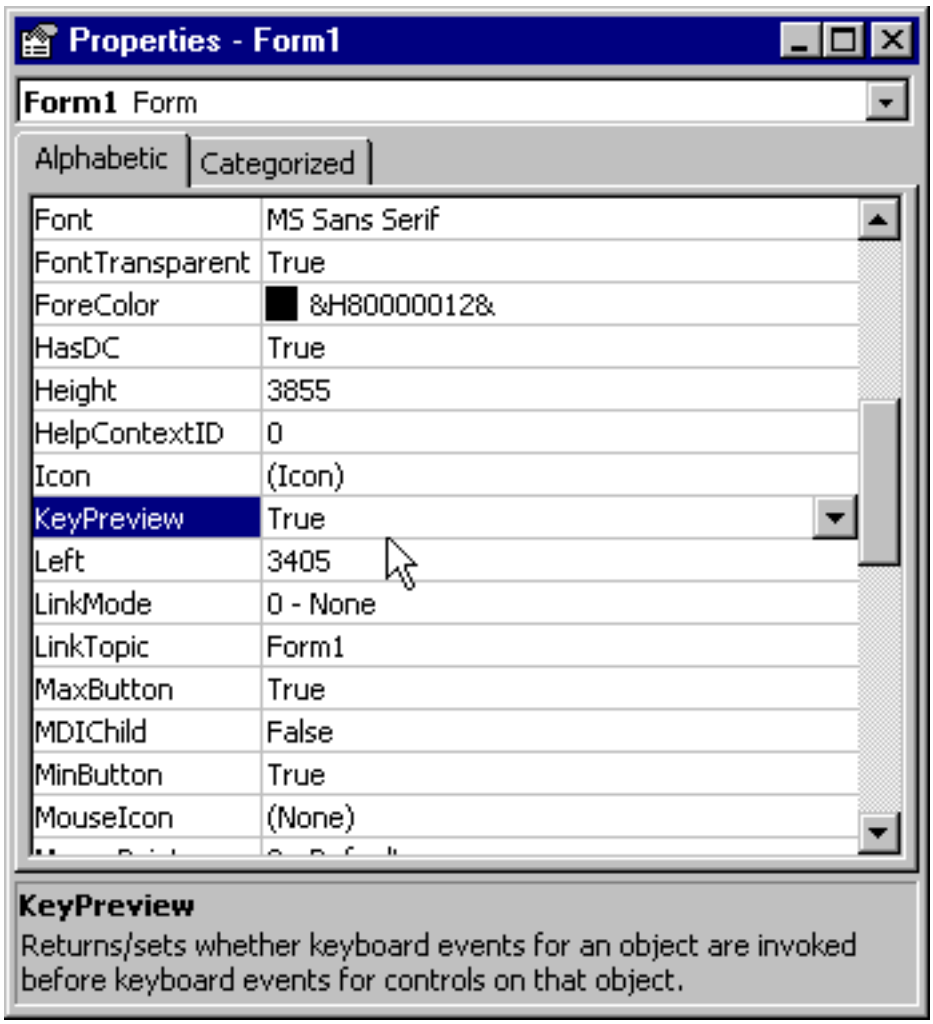
```
Private Sub cmdLetter_Click(Index As Integer)

Dim Retval As Long

Retval = PlaySound("C:\VBFILES\SOUNDS\" & cmdLetter(Index).Caption & ".
WAV", 0, 0)

End Sub
```

What I've done here is to concatenate the Caption of the Command Button that has been clicked to the file path argument of PlaySound. Now instead of evaluating 26 different case statements, this single statement can be used to play the appropriate

sound file.

Anything Else?

I had mentioned earlier that I also wanted to allow my daughter to 'type' her letters from the keyboard. To enable this in this program we need to do two things: First, we need to set the Form's KeyPreview Property to True.



Setting the KeyPreview Property of the form to true 'turns on' the KeyPress property of the form, which means anytime the user presses a key, the KeyPress Property of the Form is triggered. Now we need to write some code to deal with the keystrokes that the user generates. What will happen is that the user will press a letter, for instance 'A', which will trigger the KeyPress Event Procedure of the Form. The KeyPress Event Procedure of the form is passed an argument, equal to the ASCII value of the letter pressed on the keyboard. This is where things get a bit tricky.

Children, especially young ones, are not particularly attuned to the differences between lower case and upper case letters. For that reason, unlike the code we wrote for the Click Event Procedure of the Command Buttons, we need to be able to deal with my daughter pressing both a lower case letter and its upper case

equivalent. In other words, we need to treat the letter 'a' the same as the letter 'A'--at least as far as sounding the letter.

Let's take a look at this chart which displays the ASCII value of the letters of the alphabet, both lower and upper case, which will be 'passed' to the Form's KeyPress event…

Character	ASCII Value	Character	ASCII Value
A	65	a	97
B	66	b	98
C	67	c	99
D	68	d	100
E	69	e	101
F	70	f	102
G	71	g	103
H	72	h	104
I	73	i	105
J	74	j	106
K	75	k	107
L	76	l	108
M	77	m	109
N	78	n	110
O	79	o	111
P	80	p	112

Q	81	q	113
R	82	r	114
S	83	s	115
T	84	t	116
U	85	u	117
V	86	v	118
W	87	w	119
X	88	x	120
Y	89	y	121
Z	90	z	122

Take note of the fact that the lower case letters, in ASCII value equivalent, are 32 greater than their lower case equivalent. This fact allows us to take a lower case letter and 'convert' it to upper case by subtracting 32 from it. For instance, if the user presses the lower case 'a', its ASCII equivalent is 97, and subtracting 32 from it gives us 65, which is the ASCII equivalent of the upper case 'A'. We'll do that with this code…

```
Private Sub Form_KeyPress(KeyAscii As Integer)

Dim intIndex As Integer

If KeyAscii >= 97 And KeyAscii <= 122 Then
  KeyAscii = KeyAscii - 32
End If
```

The variable intIndex we'll see in operation in a moment. The If statement is checking to see if the value of KeyAscii is in the range of 97 through 122--if it is, we subtract 32 from it.

One more thing we'll then need to do in the KeyPress Event Procedure of the form.

We'll write code that will trigger the Click Event Procedure of one of the 26 Command Buttons, depending upon the letter that the user has pressed. Recall

that we have 26 Command Buttons, with Index values ranging from 0 to 25. If the user presses the letter 'a' or 'A', we'll wind up with a KeyASCII value of 65. At that point, we want to trigger the Click Event Procedure of the Command Button whose Index value is 0. For this reason, we'll set the value of intIndex equal to KeyASCII - 65…

```
intIndex = KeyAscii - 65
```

and then 'call' the Click Event Procedure of the Command Button whose Index value is 0 with this code…

```
cmdLetter(intIndex).Value = True
```

It's a little known fact about Visual Basic that you can trigger the Click Event Procedure of a Command Button by setting its Value property to True. Here's the full code for the KeyPress Event Procedure of the form…

```
Private Sub Form_KeyPress(KeyAscii As Integer)

Dim intIndex As Integer

If KeyAscii >= 97 And KeyAscii <= 122 Then
  KeyAscii = KeyAscii - 32
End If

intIndex = KeyAscii - 65

cmdLetter(intIndex).Value = True

End Sub
```

In fact, if we now run this program, and press either the letter 'a' or 'A', the sound of the letter 'A' will be played!

There's just one little subtle problem, though. When you press the letter 'a', which triggers the Click Event Procedure of the Command Button Control Array, the button itself never gains focus. This would be a nice touch for the user, and so we can add this line of code …

```
cmdLetter(intIndex).SetFocus
```

to the KeyPress Event Procedure of the form. This will set the focus to the appropriate Command Button. Once again, here's the full code for the KeyPress Event Procedure of the Form…

```
Private Sub Form_KeyPress(KeyAscii As Integer)

Dim intIndex As Integer

If KeyAscii >= 97 And KeyAscii <= 122 Then
  KeyAscii = KeyAscii - 32
End If

intIndex = KeyAscii - 65
cmdLetter(intIndex).SetFocus
cmdLetter(intIndex).Value = True

End Sub
```

Conclusion:

Giving your program audio capability can add impressive functionality to your program. I'm sure, based on this article, you can think of interesting enhancements to add to programs of your own.