

November, 2001

Make Them Hear You!

Tamar E. Granor

Sound can be a valuable addition to an application. Having sounds play when certain actions occur can help users to be more productive. For example, Intuit's Quicken uses the sound of a cash register to indicate that a transaction has been recorded.

However, there are some problems when using sound in applications. First, sound should never be the only medium for conveying a necessary message because users with hearing impairments and those in noisy environments may miss the message entirely. Second, even when used in addition to visual notification, the developer's choice of sounds may be inaudible to some users or simply annoying.

Fortunately, Windows provides a way for us to use sounds in applications, but give users control over the actual sounds played. There are two parts to doing this: actually playing the sounds, and giving users control over them.

Sounds in Windows

Sounds are generally implemented through .WAV files. Windows has a group of sounds defined by name and associated with particular .WAV files. For example, by default, the system sound "Asterisk" is associated with the file "Chord.WAV". Applications can play the system sounds as needed. However, individual applications can also define their own sounds and associate them with .WAV files.

Why would an application define sounds instead of just playing them directly? Because sounds that are defined are displayed in the Sounds applet, so users can change the .WAV file associated with a particular sound.

Playing Sounds in VFP apps

The easiest way to play a sound in Visual FoxPro is using SET BELL TO to specify the sound to play, then issuing ??CHR(7). However, this approach isn't particularly self-documenting. In some versions of FoxPro, it also leads to blank lines on whatever window is receiving output.

A better approach is to use the Windows PlaySound function, which is part of WinMM.DLL in the Windows API. As with other API functions, you need to declare PlaySound before you can use it:

```
DECLARE INTEGER PlaySound IN Winmm.DLL ;  
    STRING cName, INTEGER hModule, INTEGER nFlags
```

The first parameter, cName, specifies the name of a .WAV file or a Windows sound. The second parameter, hModule, can be ignored – just pass 0. The third parameter tells the function whether it's to play a file or a system sound. Pass 0 for a .WAV file or 0x100000 for a system sound. So, for example, to play the system "asterisk" sound, issue this call:

```
PlaySound( "SystemAsterisk", 0, 0x100000 )
```

To play the file Chord.WAV in the Windows\Media directory, use this call:

```
PlaySound( "C:\Windows\Media\Chord.Wav", 0, 0)
```

Registering Sounds

Information about defined sounds is stored in the Registry. Each application that defines custom sounds has a subkey under HKEY_CURRENT_USER\AppEvents\Schemes\Apps. The application's key has a value containing the name the Sounds applet uses for the application. Each define sound has a subkey under the application's key. Each of those subkeys needs at least one more subkey, named .Current, whose value is the .WAV file to play.

For example, consider an application called "Demo App" that has three custom sounds: one to play on application start-up, one to play when the application closes, and one to play when an error occurs. Table 1 shows the Registry keys we need to add to let the user choose the sounds in each case. Figure 1 shows the Registry with these keys added and some (rather strange) .WAV files specified. Figure 2 shows the registered sounds in the Sounds applet.

Table 1. Adding sounds to the Registry–To specify three sounds for the Demo App, we'd add these keys to the registry, substituting the path and file name for each relevant .WAV file. All are in HKEY_CURRENT_USER.

Key	Value
\AppEvents\Schemes\Apps\Demo	Demo App

Key	Value
\AppEvents\Schemes\Apps\Demo\Start	
\AppEvents\Schemes\Apps\Demo\Start\.Current	<.WAV to play on app start-up>
\AppEvents\Schemes\Apps\Demo\End	
\AppEvents\Schemes\Apps\Demo\End\.Current	<.WAV to play on app shut-down>
\AppEvents\Schemes\Apps\Demo\Error	
\AppEvents\Schemes\Apps\Demo\Error\.Current	<.WAV to play on error>

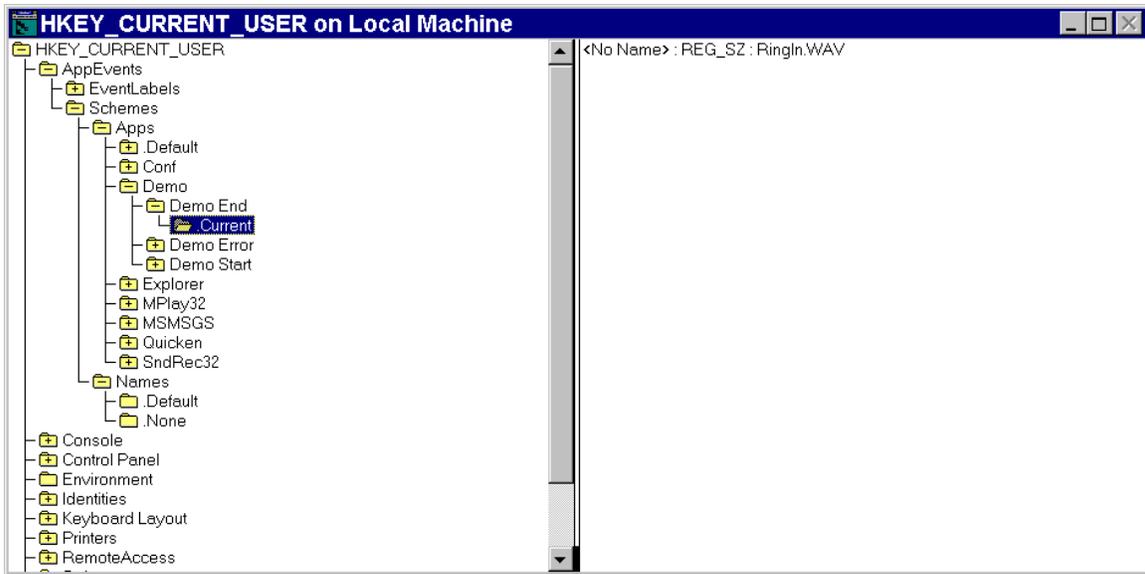


Figure 1. Custom sounds in the Registry – You can add Registry keys to define custom sounds for your applications.

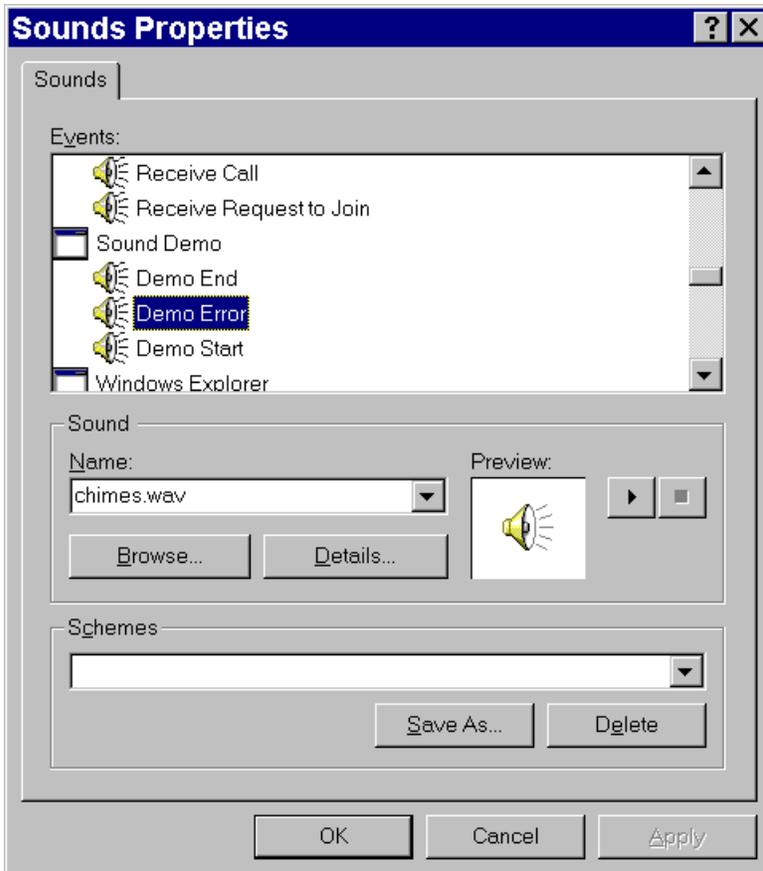


Figure 2. Registered sounds – Once sounds are defined in the Registry, they appear in the Sounds applet, where users can change the assigned .WAV files.

Note: The system sounds use one more level of indirection. Each has an internal name for the sound registered as a subkey of HKEY_CURRENT_USER\AppEvents\EventLabels; the value for each subkey is the name shown for that sound in the Sounds applet. For example, the key HKEY_CURRENT_USER\AppEvents\EventLabels\SystemAsterisk has a value of Asterisk. However, there's no requirement to do things this way and it simply adds extra steps to the process of registering and playing sounds.

Setting up Application sounds

With all this background, we're ready to store sounds for an application. To do so, we need to create the key for the application itself, and the subkeys for each sound to be registered.

Visual FoxPro comes with a class for working in the Registry that's much easier to use than direct calls to the API's registry functions. The class is Registry.VCX in VFP 6 and 7 and Registry.PRG in VFP 5. (The Registry class is documented in the VFP 6 and 7 Help files.)

The function in Listing 1 (SetSounds.PRG on this month's Professional Resource CD) registers the sounds for an application. It has 3 required parameters. The first is the internal name of the application, that is, the name to use for its registry key. The second parameter is the visible name of the application, that is, the name to show in the Sounds applet. (These two parameters can be the same.) The third parameter is an array of sounds to register. The array needs two columns: the name of the sound and the .WAV file assigned to the sound.

Listing 1. Registering Sounds—This function lets you register the sounds for an application.

```
* SetSounds.PRG
* Written by: Tamar E. Granor
* Copyright: 2001, Tamar E. Granor, Ph.D.
* Create registry entries for an app and its sounds

LPARAMETERS cInternalName, cAppName, aSoundList
* cInternalName = the internal name of the application
*               - used as the registry key
* cAppName = the name of the application to appear
*           in the Sounds applet
* aSoundList = two-column array - each row contains
*             name of a sound and the WAV file to
*             play for it.
```

```

#define HKEY_CURRENT_USER -2147483647

EXTERNAL ARRAY aSoundList

* Check parameters
ASSERT VARTYPE(cInternalName) = "C" ;
    and NOT EMPTY(cInternalName) ;
    MESSAGE "AddSounds: Must pass cInternalName"
IF VARTYPE(cInternalName) <> "C" OR EMPTY(cInternalName)
    ERROR 11
    RETURN 0
ENDIF

ASSERT VARTYPE(cAppName) = "C" and NOT EMPTY(cAppName) ;
    MESSAGE "AddSounds: Must pass cAppName"
IF VARTYPE(cAppName) <> "C" OR EMPTY(cAppName)
    ERROR 11
    RETURN 0
ENDIF

ASSERT TYPE("aSoundList[1]")="C" ;
    MESSAGE "AddSounds: Must pass array of sounds"

IF TYPE("aSoundList[1]")<> "C"
    ERROR 11
    RETURN 0
ENDIF

ASSERT ALLEN(aSoundList,2) = 2 ;
    MESSAGE "AddSounds: Array must have two columns"

IF ALLEN(aSoundList,2) <> 2
    ERROR 230
    RETURN 0
ENDIF

* If we get this far, we have parameters. Still should
* check array contents as we go.

LOCAL oRegistry, cStartKey, cNullVal, nSoundCount, nSound
LOCAL nNewSoundCount

oRegistry = NEWOBJECT("Registry",HOME()+"FFC\Registry")

WITH oRegistry
    cStartKey = "AppEvents\Schemes\Apps"
    * Create a null string
    cNullVal = ""
    cNullVal = .null.

    nNewSoundCount = 0

IF .IsKey(cStartKey, HKEY_CURRENT_USER)
    * The key we need exists. Go for it.
    * Start by creating the key for the application

```

```

IF .SetRegKey(cNullVal, cAppName, ;
  cStartKey + "\" + cInternalName, ;
  HKEY_CURRENT_USER, .t.) = 0

  * Now add the sounds, one by one
  nSoundCount = ALEN(aSoundList, 1)
  FOR nSound = 1 TO nSoundCount
    * Check that both items are provided and that
    * the file exists
    IF TYPE("aSoundList[ nSound, 1]") = "C" ;
      and TYPE("aSoundList[ nSound, 2]") = "C" ;
      and FILE(aSoundList[ nSound, 2])

      * This one looks good, so store the information
      IF .SetRegKey(cNullVal, ;
        aSoundList[ nSound, 2], ;
        cStartKey + "\" + cInternalName + "\" + ;
        aSoundList[ nSound, 1] + "\.Current", ;
        HKEY_CURRENT_USER, .t.) = 0
        nNewSoundCount = nNewSoundCount + 1
      ENDIF
    ENDIF
  ENDFOR
ELSE
  ERROR "Can't add registry key"
ENDIF
ELSE
  ERROR "Registry key does not exist"
ENDIF

ENDWITH

RETURN nNewSoundCount

```

Listing 2 shows a program (DemoSounds.PRG on the PRD) to set up the array of sounds for a demo application (the same one shown in Figures 1 and 2) and register them.

Listing 2 Setting up sound registration – This program registers the sounds for a demo application, by calling SetSounds.PRG.

```

* DemoSounds.PRG
* This program adds sounds to the Registry for a demo app.

```

```

LOCAL aSounds[3,2], cRegistryName, cVisibleName

```

```

* Set up list of sounds
aSounds[1,1] = "Demo Start"
aSounds[1,2] = "C:\WinNT\Media\RingOut.WAV"
aSounds[2,1] = "Demo End"
aSounds[2,2] = "C:\WinNT\Media\RingIn.WAV"
aSounds[3,1] = "Demo Error"
aSounds[3,2] = "C:\WinNT\Media\Tada.WAV"

```

```

* Set up app name
cRegistryName = "Demo"
cVisibleName = "Sound Demo"

nResult = SetSounds( cRegistryName, cVisibleName, @aSounds)

IF nResult < 3
    MESSAGEBOX("Problem registering sounds",48,"Accessibility Demo")
ENDIF

RETURN

```

The PRD also contains ClearSounds.PRG, which removes an application's sounds from the Registry, and DemoClearSounds.PRG, which removes the demo application's sounds.

Retrieving Application Sounds

Once the sounds for an application are registered, using them in the application is a matter of looking up the right sound in the Registry and using PlaySound to play it. Listing 3 shows a function (PlayAppSound.PRG on the PRD) that accepts an application name and a sound name and plays the specified sound, if it's found in the Registry. The function has a logical parameter that determines whether to raise an error, if the sound isn't found.

Listing 3. Playing a registered sound—This function finds and plays a specified sound.

```

* PlayAppSound.PRG
* Play a sound registered for this application
#DEFINE HKEY_CURRENT_USER -2147483647

LPARAMETERS cAppName, cSound, lErrorIfMissing
    * cAppName = the name used for the app in the registry
    * cSound = the name used for the sound in the registry
    * lErrorIfMissing = determines what happens if the
    *                   specified sound can't be found in
    *                   the registry. Defaults to .F, which
    *                   ignores the problem

* Check params
ASSERT VARTYPE(cAppName) = "C" ;
    MESSAGE "PlayAppSound: First parameter is " + ;
        "character app name"

IF VARTYPE( cAppName ) <> "C"
    ERROR 11
    RETURN .F.
ENDIF

ASSERT VARTYPE(cSound) = "C" ;
    MESSAGE "PlayAppSound: Second parameter is " + ;

```

```

        "character sound name"

IF VARTYPE(cSound) <> "C"
    ERROR 11
    RETURN .f.
ENDIF

ASSERT VARTYPE(lErrorIfMissing) = "L" ;
    MESSAGE "PlayAppSound: Third parameter is logical"

IF VARTYPE(lErrorIfMissing) <> "L"
    ERROR 11
    RETURN .f.
ENDIF

* Extract the sound
LOCAL oRegistry, cSoundFile

oRegistry = NEWOBJECT("Registry",HOME()+"FFC\Registry")

WITH oRegistry
    cStartKey = "AppEvents\Schemes\Apps"
    IF .GetRegKey("",@cSoundFile, ;
        cStartKey + "\" + cAppName + "\" + ;
        cSound + "\.Current", ;
        HKEY_CURRENT_USER) <> 0
        IF lErrorIfMissing
            ERROR "Sound not registered"
        ENDIF
        RETURN .f.
    ENDIF
ENDWITH

* Got the file name
DECLARE INTEGER PlaySound IN Winmm.DLL ;
    STRING cName, INTEGER hModule, INTEGER nFlags

PlaySound( cSoundFile, 0, 0)

RETURN

```

You can use the function as in this example, which plays the "Demo Start" sound for the Demo application:

```
PlayAppSound( "Demo", "Demo Start" )
```

On to classes

Since it's likely that you'll want to play sounds more than once for a given application, it makes sense to load all the sound information from the Registry when the application starts and play them when needed. Since this approach requires data storage as well as code, it's

a good candidate for a class. Even though registering and unregistering sounds are one-shot operations, they can also go into a class.

The PRD contains a class library called Sound.VCX with three classes:

`cusPlaySounds` plays application sounds. You need to set the `cAppInternalName` property to the name used for the application in the Registry. After that, you can call the `PlaySound` method, passing the name of the sound you want to play. The first time you call `PlaySound`, it loads all the sounds for the application into an array property.

`cusRegisterSounds` is an abstract class for registering and unregistering application sounds. To use it, subclass it, set the `cAppInternalName` and `cAppVisibleName` properties, and write code in the `SetupSounds` method to populate the `aSounds` array property with the list of sounds.

`cusRegisterDemoSounds` is a subclass of `cusRegisterSounds` configured for the Demo application of this article. To test it, instantiate it, then call the `StoreSounds` method to create the appropriate Registry keys. Use `ClearSounds` to remove the Registry keys.

The code in all three classes is quite similar to the functions earlier in the article, modified to refer to the relevant properties and methods, rather than to parameters.

Hear, hear

By defining sounds for your application and adding them to the Registry, you give users more control of your application and make it more likely that sound can be an effective part of the user interface. You also address the needs of users with auditory disabilities, bringing your application one step closer to meeting the requirements of the Americans with Disabilities Act and similar legislation around the world.