***Sound Browser* Sample Help**

Sample Description: Sound Browser

Points of Interest

- Selecting a System Drive
- Selecting a System Directory
- Searching a Directory for Files
- Handling Exceptions
- Timing Lengthy Operations
- Resizing Controls
- Disabling Controls during Lengthy Operations

Tips and Techniques

- Stripping Brackets from Directory Strings
- Declaring a Sound Player API
- Playing a Sound File

Controls

- FileComboBox
- FileListBox
- ListBox
- StopClock

For Help on Help, Press F1

Sound Browser

The Sound Browser is a finished application that can be used to quickly browse through sound files on your system. In this sample, Sound (WAV) files are located and automatically added to a browser listbox.

The Sound Browser provides controls for selecting various Drives and Directories. This may include floppy drives and CD-ROM drives as well as hard drives. Once a Drive and Directory are selected, you may click the **Search** button. Program code will search recursively through all subdirectories finding all sound (WAV) files. Once a file is located, it is added to a selection list on the Sound Browser form. When all directories have been searched, a message box will be displayed indicating the amount of time that was taken to perform the search.

To play a particular sound file, click on the corresponding filename in the list. If your system has a sound card and is connected to speakers, you should hear the sound file being played.

Selecting a System Drive

This sample uses a **FileComboBox** named **cboSelDrive** to change drives. This control specializes in selecting Drives because the **SelType** property is set to "1 - Drives." When a specific Drive is clicked, the **SelPath** property is automatically updated. Clicking on this control invokes the following Click method:

```
Sub cboSelDrive_Click()
        ' Set the current directory for the Select Directory list
        lstSelDirectory.CurrentDir = cboSelDrive.SelPath
        ' Update the Search Directory label
        lblCurDirectory.Caption = lstSelDirectory.CurrentDir
End Sub
```

This method sets the **CurrentDir** property of the **lstSelDirectory** (FileListBox) to be the selected Drive. This automatically updates the contents of the FileListBox to include all main directories on that drive. In addition, a label on the form is updated to display the current drive and directory information.

Selecting a System Directory

This sample uses a **FileListBox** named **lstSelDirectory** to change directories. This control specializes in selecting Directories because the **ShowDirs** property is set to True. To prevent Drives and Files from being included, the **ShowDrives** and **ShowFiles** properties are set to False. The property **CurrentDir** displays the current working directory. The following Double-Click event is assigned to the lstSelDirectory control:

```
Sub lstSelDirectory_DblClick()
        ' Set the Select Directory current directory to the one chosen
        lstSelDirectory.CurrentDir = lstSelDirectory.SelPath

        ' Update the Search Directory label
        lblCurDirectory.Caption = lstSelDirectory.CurrentDir

        ' Update current directory of hidden file and directory listboxes
        lstHiddenFiles.CurrentDir = lstSelDirectory.CurrentDir
        lstHiddenDirs.CurrentDir = lstSelDirectory.CurrentDir
End Sub
```

As shown in the code above, when a directory is double-clicked in the **lstSelDirectory** control, two additional FileListBox controls are updated with the **CurrentDir** information. These two controls are named: **lstHiddenFiles** and **lstHiddenDirs**. Each of these controls were embedded directly in the Property Editor. When this is done, they are not displayed on the form. The lstHiddenFiles FileListBox has its **ShowFiles** property set to True, while the lstHiddenDirs FileListBox has its **ShowDirs** property set to True.

This application is basically organizing the files and directories in two FileListBoxes each time a directory is clicked by the user. These two "hidden" FileListBoxes will play a key role as we recursively search through sub-directories. It is important to note that the lstHiddenFiles control has its **Filter** property set to ".wav". This means that only Sound files (.WAV) will be displayed in this list. All files shown in this list will end up in the **lstWaveFiles** ListBox for user selection.

Searching a Directory for Files

The actual file search is triggered by the **Search** Button through the **btnSearch_Click** method. In this method, a Sub procedure called **GenerateWaveList** is called and given the name of this current directory.

The way this routine works is that the contents of the "hidden" list, lstHiddenDirs, is looked at line by line. For each sub-directory that is found, additional sub-directories are searched for. This recursive search continues down the directory structure until no additional sub-directories are found. At this point, since the lstHiddenFiles FileListBox is updated at the same time as lstHiddenDirs, all Sound (WAV) files would appear in this list. Each time sound files are present, they are extracted and added to the lstWaveFiles ListBox.

Handling Exceptions

When the **Search** Button is clicked, a recursive search operation through a Directory structure is started. Depending on the directory and the size of your disk drive, this search may take a matter of seconds. In order to be able to interrupt this search operation, the **Caption** on the **Clear** button is changed to "Cancel" during the search. If the "Cancel" button is clicked during the search operation, an AbortFlag property is set to True.

The GenerateWaveList Sub procedure contains the following Try/Catch code:

```
' Initiate the recursive search for matching files
Try
        GenerateWaveList lstSelDirectory.CurrentDir
Catch AbortFlag()
        InfoBox.Msg("Search operation canceled.")
End Try
```

If the Cancel button is clicked during the search operation, the following program code inside the GenerateWaveList Sub procedure detects the True status of the AbortFlag and "Throws" an exception.

```
AbortFlag = 0
' Free up CPU to update display
  Application.DoEvents()

  ' Check the AbortFlag status
  If (AbortFlag) Then Throw AbortFlag()
```

In order to allow an interruption, an Application.DoEvents statement was added to this program code. This frees up the CPU and processes any outstanding events, such as a Click event, from the Cancel button. If the user clicks the Cancel button before the search operation has completed, the AbortFlag property is set to True and is detected by the above check, then the **Throw** event is caught by the **Catch** located in the original routine. At this point, a message can be posted and the application restored to a non-search condition.

Timing Lengthy Operations

Some application functions may take a little time to complete. In this sample, depending on the directory you are searching and the size of your drive, the search may take several seconds. To show you how to time a lengthy operation, a **StopClock** object has been embedded in this sample form named "**tmrStopWatch**."

When the **Search** Button is clicked, the **State** property of the tmrStopWatch is set to 3, which starts the timer running. The following options exist in the State property:

State	**Action**
0	Reset
1	Stopped
2	Paused
3	Running

When the search operation is complete, the State property is set to 0, which stops the timer. The total time is recorded in the **ElapsedTime** property. After the search has finished, a Message Dialog is posted indicating the total time that had elapsed during the search operation. In addition, the total number of files that were located is also determined in the following code example:

```
total_time = tmrStopWatch.ElapsedTime

file_count = lstWaveFiles.ListCount

InfoBox.Msg(file_count & " sound files located in " & total_time & " time.")
```

This is one way that lengthy operations can be timed. It is up to you to decide what action you want your application to carry out based on the elapsed time. In this sample, the time is merely posted in a message dialog.

Resizing Controls

The sample form contains a Resize method that is trigged each time the form is resized. The purpose of the Resize method is to allow the lstWaveFiles ListBox to grow wider and higher as the form is made wider and higher. In addition, the Directory FileListBox, the Directory Label, and the two Buttons at the bottom right of the form must also be automatically moved and resized when the form size is changed.

The **Resize** method contains program code that automatically sizes each of the necessary controls on the form. Minimum gaps between controls are defined to keep controls from overlapping each other. After all the controls are moved and resized, the Resize method invokes a **Refresh** method to redraw the form and its controls.

Disabling Controls during Lengthy Operations

When an application operation may take a few seconds to complete, it is a good idea to disable controls on the form to prevent unwanted clicks while the operation, such as the search operation in this sample, is running. This can be done by disabling various controls on the form by setting their **Enabled** property to False during the search operation, and setting the property back to True once the search is completed. Below is an example of how to disable controls on the form:

```
lstWaveFiles.Enabled = 0
lstSelDirectory.Enabled = 0
cboSelDrive.Enabled = 0
btnClear.Caption = "Cancel"
btnSearch.Enabled = "False"
```

Notice that setting the Enabled property to False will disable the control. Likewise, setting the Enabled property to True will enable or activate the control. Another way to let the user know that a lengthy operation is in progress is to change the Mouse cursor to an "Hour glass" during the operation, then reset it to the standard "Arrow pointer" when the task is finished.

Stripping Brackets from Directory Strings

As this program code browses through the directory structure searching for files, the names of directories presented in the lstHiddenDirs contain brackets [] around each of the directory names. Since this program code only wants to use the names of the directories, the following function is used to strip the brackets from directory names:

```
Function StripBrackets(dir_name As String) As String
    Dim left_char As String
    Dim right_char As String
    Dim string_size As Integer

    ' Save the first character
    left_char = Left(dir_name, 1)
    ' Save the last character
    right_char = Right(dir_name, 1)

    ' If brackets enclose the string, remove them
    If left_char == "[" And right_char == "]" Then
        string_size = Len(dir_name) - 2
        StripBrackets = Mid(dir_name, 2, string_size)
    Else
        ' Pass back the original string since it has no brackets
        StripBrackets = dir_name
    End If

End Function
```

This is just a simple technique that uses the **Mid** and **Len** language commands to remove unwanted characters from a text string.

Declaring a Sound Player API

The capability for playing sound files is assisted by a Windows API (Application Program Interface). In this Sound Brower application, an object named AudioPlayer has been declared to assist in the playing of sound files. The following statement was used to declare the sound player function:

```
Declare Function sndPlaySoundA Lib "Winmm" (ByVal file As String, ByVal opt As Long) As Long
```

Playing a Sound File

You can play a sound file by clicking on a specific sound file in the lstWaveFiles ListBox. The following click method is executed when a sound file is clicked:

```
Sub lstWaveFiles_Click()
  Dim option As long
  Dim result As long
  Dim sound_file As String

  ' Set a variable to be the name of sound file including absolute path
  sound_file = lstWaveFiles.ItemString(lstWaveFiles.ListIndex)

  ' Play the selected sound file
  result = AudioPlayer.sndPlaySoundA(sound_file, option)

End Sub
```

The function **sndPlaySoundA** is passed to the sound file, which contains the name of the file as well as the directory path. This function executes a corresponding API function to play the selected sound file.