

Bio 595

Computers in

Biomedical Research

Class notes set 5

Fall 2003

Accessing Disk Files

"File" = a collection of data, (either text or database format.

**Files with the same filename must reside on different disks or in different directories
Easiest to use are text or ASCII files.**

Other files are encoded in binary (machine-readable only) formats (we won't use these).

You must direct VB to the exact location of the file you want to access.

Do this using the File Open dialog box (allows selection of file to be accessed from any drive and subdirectory).

Open File

The format is

**Open strFileName [For Mode] As
(([#]intFileNumber).**

The FileName given must reside on the default drive unless the full pathname is specified.

Use the CurDir() function to determine the current directory, then stick this string into the full pathname.

“Mode” specifies what you wish to do with the accessed data.

Open File, continued

Modes:

Append (add new data to the end of a file).

Input (to read data from the file - that is, input to your program).

Output (write data into the file - that is, output from your program).

FileNumber is an integer between 1 and 255.

FileNumber (or channel) remains associated with the data file until that file is closed.

VB can have multiple channels open at one time.

Close File

Finishes the writing of any file data into the file (from the buffer), and then releases file number for subsequent use.

"Close" without arguments closes all open files.

The format is

Close [#]intFileNumber] [,...,[#intFileNumber]

Delete File

This "Kill" statement permanently erases a file from the disk.

The format is:

Kill "c:\Dat\MyData.DAT"

This erases MyData.DAT in the file folder on C called Dat.

Write to a File

The **Write#** command writes any type of data to a file opened in "Output" or "Append" modes.

This works on strings and numbers in all combinations.

The format is:

Write #intFileNumber [,ExpressionList]

Write to a File

Here `intFileNumber` is associated with file opened for output. If no values are given, a CR LF (carriage return and line feed) are provided.

Items are separated by commas; the end of each written line is terminated with a CR LF.

Quotation marks are added around all strings in the file.

Logical values are bracketed by pound sign (e.g. `#True#`).

A `Write #` line is ended with a semicolon to get next `Write#` to continue on the same line of the data file.

Example: Write to a File

**Write #3, intIDno, intAge, strPatName, strDrName,
strBalance**

**This will record a single line of data to the file that would
look like**

1047,31,"Sam Jones","J. D. Smith, MD", "2344.75"

**Such a "Write #" statement could be within a
For...To...Next... loop where each of the integer and
string variables could be subscripted with an indexing
variable like i (e.g. intIDno(i), etc.).**

**Write# will add to the end of a file if the file is opened in
Append mode.**

**Otherwise, opening in Output mode will cause the data
to overwrite existing data in the file.**

Input Data to a File

Input# reads data into your program from a stored file.

The format is

Input #intFileNumber [,EprressionList]

This is the reverse of the **write#** statement.

You must match incoming data with the appropriate data type.

Attempts to read more data into your program than actually exists in the file will generate an error message. To avoid this...

EOF() Function

Use the EOF() function to test for the end of the file.

The format is EOF(intFileNumber).

This function returns True if at the end of the file, or False if not.

A good program design is to use Do Until the end of file is reached:

Do Until (EOF(intLineNumber) = True)

Use a counter in the loop to 'remember' the number of data values retrieved.

Line Input#

This reads data from a file into the program in the form of a single string variable.

The line of data is expected to end with CR LF (the way most file records end).

Then the data can be parsed.

Format Statement

The Format statement sets numeric or string values to the desired specification (such as two decimal places to the right, etc.).

The format for the format statement is

Format (expression, strFormat)

Example:

```
txtEnding.Text=Format(calibration*scalefactor, "#0.00")
```

Ideas for Error Checking

To insure correct input of values, you can use a message box to report any problems on the screen.

You could convert entered text box text into numbers, check ranges of entered values, then use the message box to tell user of any errors.

Set the focus on control with bad value so the user can see where the error is.

Use an error checking function to test values, call it `ErrorCheck()`; if it is set (= 1), exit the subroutine for the calculation.

Ideas for Error Checking

Add a module to a project for error checking

Add a module (default name module1, filename module1.bas)

Check all text box values for correct range, return a 1 value for ErrorCheck() function if any errors are found:

Public Function ErrorCheck() As Integer

(various If...Then... tests to determine if user input is in range)

End Function.

List Boxes

Call up the List Box control from the Tools | Options | Editor Format selection on the pulldown menu.

Size the box on the form appropriately for the width of the data; scrollbars are provided automatically.

The box can have 1, 2 or 3 columns (set via Columns property).

List Box Properties

BackColor Specifies the list box's background color.

Columns Determines the number of columns. If 0, the list box scrolls vertically in a single column. If 1 or more, the list box items appear in the number of columns specified (one or more columns) and a *horizontal scrollbar* appears so you can see all the items in the list.

ForeColor Specifies the list box's text color.

Height Indicates the height of the list box in twips.

List Box Properties

IntegralHeight Determines whether the list box can display partial items, such as the upper half of an item that falls toward the bottom of the list box.

List Holds, in a drop-down property list box, values that you can enter into the list box at design time. You can enter only one at a time, and most programmers usually prefer to initialize the list box at runtime.

List Box Properties

MultiSelect The state of the list box's selection rules. If 0 -None (the default), the user can select only one item by clicking with the mouse or by pressing the Spacebar over an item. If 1 -Simple, the user can select more than one item by clicking with the mouse or by pressing the Spacebar over items in the list. If 2 -Extended, the user can select multiple items using Shift+click and Shift+arrow to extend the selection from a previously selected item to the current one. Ctrl+click either selects or deselects an item from the list.

List Box Properties

Sorted Determines whether the list box values are automatically sorted. If False (the default value), the values appear in the same order in which the program added the items to the list.

Style Determines whether the list box appears in its usual list format or with check boxes next to the selected items.

ListBox Methods

List box “methods” let user identify selections, add items or remove items from list.

AddItem Adds a single item to the list box.

Clear Removes all items from the list box.

List A string array that holds items from within the list box.

ListCount The total number of list box items.

RemoveItem Removes a single item from the list box.

Example of ListBox methods

lstOneCol.AddItem “Joseph” adds that name to list.

AddItem is used for addition to the list

RemoveItem is used to delete items from the list

These can be subscripted, counting from the first item as element 0

Clear removes all list items

List Box Methods

Items in a list can be assigned to variables
(example: `strVar1 = lstOneCol.list(3)`)

Use `ListCount` to determine the number of items in a list (example: `intNum = lstOneCol.ListCount`); such a value could then be used as the end value in a `For...To...Next` loop.

“`Selected`” tells whether an item has been chosen by the user (if `True`, the item was selected).

`MultiSelect` permits more than 1 item to be selected.

Combo Boxes

These behave like List Boxes but also permit addition of items to the list

The three types of combo Boxes include

- Drop-down; a single line with a down arrow;**
- simple, looking like a list box, but allowing items to be added**
- drop-down list box, a single line equivalent of list box, with no ability to add items.**

These three can be set via Style combo box property.

Combo Box Properties

BackColor The combo box's background color.

ForeColor The combo box's foreground text color.

Height The height in twips of the closed combo box.

IntegralHeight Determines whether the combo box can display partial items, such as the upper half of an item that falls toward the bottom of the combo box.

List A drop-down property list box in which you can enter values into the combo box at design time. You can enter only one at a time, and most programmers prefer to initialize the combo box at runtime.

Combo Box Properties

Sorted Determines whether the combo box values are automatically sorted. If False (the default value), the values appear in the same order in which the program added the items to the combo box.

Style Determines the type of combo box your application needs. If 0 - DropDown Combo, the combo box is a drop-down combo box. If 1 - simple Combo, the combo box turns into a simple combo box that remains open to the height you use at design time. If 2 - DropDown List, the combo box turns into a drop-down list box that remains closed until the user is ready to see more of the list.

Combo Box Caution

Items entered via a combo box don't automatically get added to the list without some necessary code.

**For example, use `cboBox.AddItem cboBox.Text`.
Or, you could add a command button to add the user entry.**

Data Arrays

Arrays are lists of subscripted variables.

Example: DataSet(5) is the sixth saved value in an array of single precision numbers from Dataset.

Use the Option Base 1 statement to begin subscripts with (1) instead of (0).

Use Dim or Public to declare array variables, specifying the array size in parentheses.

Example: Dim sngDataSet(10) As Single.

Subscript indexing makes it easy to process all the data in an array.

Data Arrays

In this simple example, we find the average value of 5 elements in a data array Dens(1) through Dens(5):

AveDens = 0

For i = 1 to 5

AveDens = AveDens + Dens(i)

Next i

AveDens = AveDens/5

Data Arrays

Arrays can be a list (a one dimensional array) like `SpecData(t)` where the index variable `t` ranges from 0 to 99 – a list of 100 values, `SpecData(0)` the first and `SpecData(99)` the last, or from `SpecData(1)` to `SpecData(100)`

Arrays can be a table (a two dimensional array) like `Dens(w,t)` where the first index variable `w` represents a set of measurements for trials 1 to 10 (10 columns) and the second index variable `t`, for time of the measurement, ranges from 1 to 100

Data Arrays

Arrays can be multidimensional (3 dimensions or higher)

For example, the declaration “Dim Expt(5,10,100) As Single” would create a 3-dimensional array of data referenced as Expt(v,w,t) where v represented one of 5 or 6 concentrations of a drug in an experiment, from 0 or 1 to 5, w the trial run (either from 0 or 1 to 10) and t the time of the measurement, from 0 or 1 to 100