

Bio 595

Computers in

Biomedical Research

Class notes set 4

Fall 2003

Putting Code into Visual Basic

A VB program consists of forms, controls placed on the forms, and VB code

The code manipulates data and performs I/O (input and output operations)

Some code is a lot of small event procedures in form modules and other code in standard (non-form related) modules.

Sometimes a program might have no form (the form's visible = false).

Definitions

A "form module" is the module file containing one or more forms used in an application, and the code (e.g. event procedures) that goes with each form.

A "standard module" is a file holding code not connected to a form.

Data Types in VB

Boolean: A data type that takes on one of two values only: True or False. True and False are Visual Basic reserved words, meaning that you cannot use them for names of items you create.

Byte: Positive numeric values without decimals that range from 0 to 255.

Currency: Data that holds dollar amounts from -\$922,337,203,685,477.5808 to \$922,337,203,685,477.5807. The four decimal places ensure that proper rounding can occur. VB respects the Windows International settings and adjusts currency amounts according to your country's requirements. Never include the dollar sign when entering Currency values.

Data Types in VB

Date: Holds date and time values. The date can range from January 1 100, to December 31, 9999. (In the years following 9999, people will have to use something other than Visual Basic!)

Decimal: A new data type not yet supported in Visual Basic except in a few advanced situations. The Decimal data type represents numbers with 28 decimal places of accuracy.

Double: Numeric values that range from $-1.79769313486232E+308$ to $1.79769313486232E+308$. The double data type is often called double-precision.

Integer: Numeric values with no decimal point or fraction that range from $-32,768$ to $32,767$.

Data Types in VB

Long: Integer values with a range beyond that of Integer data values. Long data values range from -2,147,483,648 to 2,147,483,647. Long data values consume more memory storage than integer values, and they are less efficient. The Long data type is often called long integer.

Object: A special data type that holds and references objects such as controls or forms.

Single: Numeric values that range from $-3.402823E+38$ to $3.402823E+38$. The Single data type is often called single-precision.

Data Types in VB

String: Data that consists of 0 to 65,400 characters of alphanumeric data.

Alphanumeric means that the data can be both alphabetic and numeric. String data values may also contain special characters such as ^, %' and @. Both fixed-length strings and variable-length strings exist.

Variant: Data of any data type and used for control and other values for which the data type is unknown.

Scientific Notation

The letters E and D designate exponent and double precision exponent respectively.

Literals are by nature constants

Strings are literals of text in quotation marks

“” is an empty, or null string

Time and date are embedded as literals between pound (#) signs.

Use data type suffix characters to specify appropriate type: “&” for long, “!” for single, “#” for double, and “@” for currency.

Variables

These are named locations in computer memory that hold data.

A variable must be declared, identifying the data type it is to hold.

Use the “Dim” statement to declare variables.

The format is *Dim VarName as DataType*.

Use Option Explicit statement to tell VB you will declare variables and their data types (otherwise, VB assumes variant data type for variables).

Variables

Dim in an event procedure means the variable is local.

Dim in a general module makes the variable global.

“Public” instead of Dim makes the variable(s) visible in every module of the project.

It is recommended (but not required) that the standard 3-letter prefixes be used in variable names, identifying the data type.

Variables

Prefix	Data Type	Example
bln	Boolean	blnIsOverTime
byt	Byte	bytAge
cur	Currency	curHourlyPay
dtm	Date	dteFirstBegan
dbl	Double	dblMicroMeasurement
int	Integer	intCount
lng	Long Int	lngStarDistance
obj	Object	objSoundClip
sng	Single	sngYearSales
str	String	strLastName
var	Variant	varControlValue

Assignment Statement

This puts data into a variable.

The format is VarName = Expression (LET implied)

The expression here can be a literal, variable, or math expression.

You must match data type and declared variable type.

Example: sngTemp = 42.1 is valid, but sngTemp = "Hot!" is not.

Note that text boxes or command buttons have properties which, as variables, can be reset by code during program execution.

Math Operators

**+ (add), - (subtract), * (multiply), / (divide
^ (raise to power), & (concatenate strings)**

**Precedence: first ^, then * and /, then + and -,
from left to right.**

Use parentheses to avoid ambiguity

Example of a string concatenation:

**strFullName = strFirstName & " " & strLastName
(the & " " inserts a space between the first and
last names).**

Functions

These include intrinsic or built-in functions such as common math operations, string manipulations, and I/O operations.

Functions accept arguments, and return a result for use by the program.

Arguments are the data given to a function.

Two functions: MsgBox() and InputBox().

The presence of the parentheses indicates the name identifies a function, not a variable.

A message box gives the user output information

An input box asks the user for input information

The MsgBox() Function

Assign these functions to an integer variable so that the button clicked by a user can be saved.

The format is:

```
an IntVariable = MsgBox( strMsg [intType] [ , strTitle])
```

Here, **strMsg** is a text string (either variable or constant), the message to be displayed;

intType is the optional value describing desired options in the box;

strTitle is the Message box title - if omitted, the Project's name is used.

Buttons displayed in a message box

Named Literal	Value	Description
vbOKOnly	0	Displays OK button
vbOKCancel	1	Displays OK, Cancel buttons
vbAbortRetryIgnore	2	Displays Abort, Retry, and Ignore buttons.
vbYesNoCancel	3	Displays the Yes, No, and Cancel buttons.
vbYesNo	4	Displays Yes and No buttons
vbRetryCancel	5	Displays Retry, Cancel buttons

Icons displayed in a message box

Named Literal	Value	Description
vbCritical	16	Displays Critical Message icon
vbQuestion	32	Displays Warning Query icon
vbExclamation	48	Displays Warning Message icon
vbInformation	64	Displays Information Message icon

Default buttons displayed in a message box

Named Literal	Value	Description
vbDefaultButton1	0	The first button is the default
vbDefaultButton2	256	The second button is the default
vbDefaultButton3	512	The third button is the default

MsgBox() function, continued

The options selected use the intType value in the MsgBox() function, and determine whether an icon is displayed, and whether the message box is application specific or system specific.

If application specific, the box must be answered before the program proceeds.

If system specific, the box must be closed even to switch to another Windows application.

Example of a MsgBox function:

```
intPress = MsgBox("Are you ready for the  
report?", vbQuestion + vbYesNoCancel,  
"Report Request")
```

MsgBox () return values

These integers report back which button was clicked.

Named	Value	Description
vbOK	1	The user clicked the OK button.
vbCancel	2	The user clicked the Cancel button.
vbAbort	3	The user clicked the Abort button.
vbRetry	4	The user clicked the Retry button.
vbIgnore	5	The user clicked the Ignore button.
vbYes	6	The user clicked the Yes button.
vbNo	7	The user clicked the No button.

Comments

Remarks added to a program constitute an essential part of the documentation that the program's author must provide to the user.

Critical information to be provided include the program's author, date, what the general program is to accomplish, the intent or goal of each procedure, and any explanations deemed necessary to explain difficult logical functioning of the program (the program's algorithm, or strategy for solving the problem).

Begin remark lines with the Rem statement, or with an apostrophe.

These can be added to the right of code lines on the same line.

InputBox testing

The format of this function is

```
strVariable = InputBox( strprompt [, (strTitle) [,  
    strDefault] [, intXpos, intYpos])])
```

Here **strPrompt** is inside the displayed box; **strTitle** is the title inside the input box's title bar, **strDefault** is a default string value VB displays for the default answer (which can be changed by the user). **IntXpos** and **IntYpos** give the input box location on the form, in twips from top and left edges; if omitted, box is centered on the form.

Example:

```
strCompName = InputBox("What is the name of the  
    Company?", "company request, "XYZ, Inc.")
```

Comparison Operators

These yield “true” or “false” results in comparing data values with one another.

Comparison operators can compare variables, literals, control values, etc.

Comparisons work both with numeric and alphanumeric values, allowing comparisons of strings.

Strings are equated to their ASCII code values.

Thus, “B” > “C” (comparing ASCII codes 65 and 66) and “Smith, K.” < “Smith, L.”

Comparisons must be of consistent data types or a mismatch error will be generated.

Comparison Operators

- > example: IblLab.Caption > Goal** The *greater than* operator returns True if the value on the left side of > is numerically or alphabetically greater than the value on the right.
- < example: Bill < 2000.00** The *less than* operator returns True if the value on the left side of < is numerically or alphabetically less than the value on the right.

Comparison Operators

- = example: Age = Limit** The *equal to* operator (sometimes called the *equal operator*) returns True if the values on both sides of = are equal to each other.
- >= example: FirstName >= "Mike"** The *greater than or equal to* operator returns True if the value on the left side of >= is numerically or alphabetically greater than or equal to the value on the right.

Comparison Operators

<= Num c= lblAmt.Caption The *less than or equal to* operator returns True if the value on the left side of **c=** is numerically or alphabetically less than or equal to the value on the right.

<> txtAns.Text <> "Yes" The *not equal to* operator returns True if the value on the left side of **<>** is numerically or alphabetically unequal to the value on the right.

If statement

This uses the comparison operator results to test data.

“IF” will execute subsequent lines of code if the result is true, or other lines of code if the result is false.

The format is:

IF comparisonTest THEN

One or more lines of VB code (a “block” of code)

ELSE

One or more lines of different VB code

END IF

ELSE here is optional and is code executed if the evaluation result is False.

Either the first block of code or the second executes (they’re “mutually exclusive”)

Logic Operators

And If $(A > B)$ And $(C < D)$ Produces True if both sides of the And are true. Therefore, A must be greater than B and C must be less than D. Otherwise, the expression produces a false result.

Or If $(A > B)$ Or $(C < D)$ Produces True if either side of the Or is true. Therefore, A must be greater than B or C must be less than D. If both sides the Or are false, the entire expression produces a false result.

Not If `Not (strAns = "Yes")` Produces the opposite true or false result. Therefore, if strAns holds Yes ", the Not turns the true result to false.

Select Case

This statement allows multiple choices to be made; it simplifies otherwise deeply nested If-Then-Else statements. Its format is

Select Case *expression*

Case *value*

One or more VB statements

Case *value*

One or more VB statements

Case *value*

One or more VB statements etc.

End Select

Here *expression* must evaluate to a numeric or string value.

Another format of Select Case

Select Case *expression*

Case is *relation*:

One or more VB statements

Case is *relation*:

One or more VB statements

Case Else:

One or more VB statements

End Select

Instead of comparing Expression values for an exact case match, this second format allows separate comparison tests.

Still another Select Case option

Select Case *expression*

Case expr1 to expr2:

One or more VB statements

Case expr1 to expr2:

One or more VB statements

Case expr1 to expr2:

One or more VB statements

End Select

Here the Case lines are given a range of values instead of one exact match.

Do While loop

Like If...Then..., Do While works with the six comparison expressions listed earlier.

The format is

Do While (comparison test)

Block of VB statements

Loop

The loop is repeated as long as the comparison test is true.

Something in the block of VB code had better change a variable value within the comparison test, or an “infinite loop” will occur.

As soon as the comparison test becomes false, the loop terminates.

The loop terminates as soon as the test is false, so the loop will never execute if the comparison is initially false.

Do While loop executes as long as comparison test is true

```
Dim strAge As String
Dim intAge As Integer
Dim intPress As Integer
' Get the age in a string variable
strAge = InputBox("How old are you?", "intAge Ask")
' Check for the Cancel command button
If (strAge = "") Then
End ' Terminates the application
End If
' Cancel was not pressed, so convert Age to integer
' The Val() function converts strings to integers
intAge = Val(strAge)
' Loop if the age is not in the correct range
Do While ((intAge < 10) Or (intAge > 99))
' The user's age is out of range
intPress = MsgBox("Your age must be between " & _
"10 and 99", vbExclamation, "Error!")
strAge = InputBox("How old are you?", "Age Ask")
' Check for the Cancel command button
If (strAge = "") Then
End ' Terminate the program
End If
intAge = Val(strAge)
Loop
```

Do Until loops until comparison test becomes true

```
Dim strAge As String
Dim intAge As Integer
Dim intPress As Integer
' Get the age in a string variable
strAge = InputBox("How old are you?", "Age Ask")
' Check for the Cancel command button
If (strAge = "") Then
End ' Terminate the program
End If
' Cancel was not pressed, so convert Age to integer
intAge = Val(strAge)
' Loop if the age is not in the correct range
Do Until ((intAge >= 10) And (intAge <= 99))
' The user's age is out of range
intPress = MsgBox("Your age must be " & _
"between 10 and 99", vbExclamation, "Error!")
strAge = InputBox("How old are you?", "Age Ask")
' Check for the Cancel command button
If (strAge = "") Then
End ' Terminate the program
End If
intAge = Val(strAge)
Loop
```

Using the Do...Loop While to check the comparison at the bottom of the loop

```
Dim strAge As String
Dim intAge As Integer
Dim intPress As Integer
Do
strAge = InputBox("How old are you?", "Age Ask")
' Check for the Cancel command button
If (strAge = i"") Then
End ' Terminate program
End If
intAge = Val(strAge)
If ((intAge < 10) Or (intAge > 99)) Then
' The user's age is out of range
intPress = MsgBox("Your age must be between " & _
    '10 and 99", vbExclamation, "Error!")
End If
Loop While ((intAge < 10) Or (intAge > 99))
```

For...To...Next

Rem- Add the numbers from 1 to 100.

IntSum = 0

For intNumber = 1 To 100

IntSum = IntSum + intNumber

Next intNumber