Training Manual

FOR

Microsoft®

Visual Basic Programming

Compiled by **Technology Ed**

April 2012

CHAPTER-4: DEBUGGING

This chapter describes the debugging process in Visual Basic. Only through lots of use and practice can you become a proficient debugger. The following are the general guidelines:

- Keep the code simple. Many times, you only have one or two bad lines of code. And you, knowing your code best, can quickly narrow down the areas with bad lines. Do not set up some elaborate debugging procedure if you have not tried a simple approach to find your error(s) first.
- A tried and true approach to debugging can be called Divide and Conquer. If you are not sure where your error is, guess somewhere in the middle of your application code. Set a breakpoint there. If the error has not shown up by then, you know it is in the second half of your code. If it has shown up, it is in the first half. Repeat this division process until you have narrowed your search.
- Be careful when you first design and write your application to minimize searching for errors later.

4.1 Overview of Debugging

Visual Basic provides an excellent set of debugging tools to aid in this search for, and elimination of, logic errors. These are errors that don't prevent an application from running, but cause incorrect or unexpected results.

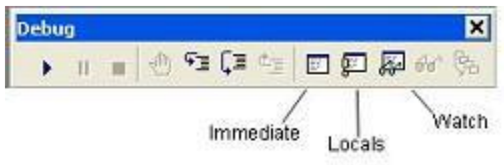
Debugging a code is an art, not a science. There are no prescribed processes that you can follow to eliminate all logic errors in your program. The usual approach is to eliminate them as they are discovered.

This section describes the debugging tools available in the Visual Basic environment (several of which appear as buttons on the toolbar) with an example. You, as the program designer, must select the debugging approach and tools you feel most comfortable with.

In Visual Basic, the interface between your application and the debugging tools is through three different debug windows:

- Immediate Window
- Locals Window
- Watch Window

You can access these windows from the View menu (the Immediate Window can be accessed by pressing Ctrl+G). Or, you can select from the Debug Toolbar (accessed using the Toolbars option under the View menu):



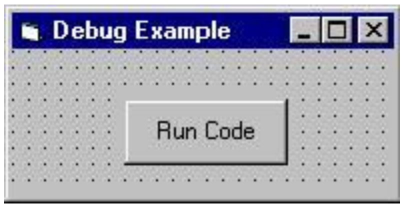
All debugging using the debug windows is done when your application is in break mode. You can enter break mode by setting breakpoints, pressing Ctrl+Break, or the program will go into break mode if it encounters an untrapped error or a Stop statement.

Once in break mode, the debug windows and other tools can be used to:

- Determine values of variables
- Set breakpoints
- Set watch variables and expressions
- Manually control the application
- Determine which procedures have been called
- Modify the values of variables and properties

Debugging - Example

1. Unlike other examples, you can do this one as a group. It will be used to demonstrate the use of debugging tools.
2. The example simply has a form with a single command button. The button is used to execute some code.



3. The code attached to this button’s Click event is a simple loop that evaluates a function at several values.

```
Private Sub Command1_Click()

Dim X As Integer, Y As Integer

X = 0

Do

Y = Fcn(X)

X = X + 1

Loop While X <= 20

End Sub
```

This code begins with an X value of 0 and computes the Y value using the general integer function Fcn. It then increments X by 1 and repeats the Loop. It continues looping While X is less than or equal to 20. The function Fcn is computed using:

```
Function Fcn(X As Integer) As Integer

Fcn = CInt(0.1 * X ^ 2)

End Function
```

Admittedly, this code doesn't do much, especially without any output, however it makes a good example for looking at debugger use.

4.2 Using the Debugging Tools

There are several debugging tools available for use in Visual Basic. Access to these tools is provided with both menu options and buttons on the Debug toolbar. These tools include breakpoints, watch points, calls, step, and so on. The simplest tool is the use of direct prints to the immediate window.

Printing to the Immediate Window:

You can print directly to the immediate window while an application is running. Sometimes, this is all the debugging you may need. A few carefully placed print statements can sometimes clear up all logic errors, especially in small applications.

To print to the immediate window, use the Print method:

```
Debug.Print [List of variables separated by commas or semi-colons]
```

Debug.Print Example:

1. Place the following statement in the Command1_Click procedure after the line calling the general procedure Fcn and run the application:
   ```
   Debug.Print X; Y
   ```
2. Examine the immediate window. Note how, at each iteration of the loop, the program prints the value of X and Y. You could use this information to make sure X is incrementing correctly and that Y values look acceptable.
3. Remove the `Debug.Print` statement.

Breakpoints:

In the above examples, the program ran to completion before we could look at the debug window. In many applications, we want to stop the application while it is running, examine variables and then continue running. This can be done with breakpoints.

A breakpoint is a line in the code where you want to stop (temporarily) the execution of the program that is force the program into break mode. To set a breakpoint, put the cursor in the line of code you want to break on. Then, press <F9> or click the Breakpoint button on the toolbar or select Toggle Breakpoint from the Debug menu. The line will be highlighted.

When you run your program, Visual Basic will stop when it reaches lines with breakpoints and allow you to use the immediate window to check variables and expressions. To continue program operation after a breakpoint, press <F5>, click the Run button on the toolbar, or choose Start from the Run menu.

You can also change variable values using the immediate window. Simply type a valid Basic expression. This can sometimes be dangerous, though, as it may change program operation completely.

Breakpoint Example:

1. Set a breakpoint on the X = X + 1 line in the sample program and then run the program.
2. When the program stops, display the immediate window and type the following line:
   ```
   Print X;Y
   ```
3. The values of these two variables will appear in the debug window. You can use a question mark (?) as shorthand for the command Print, if you'd like. Restart the application. Print the new variable values.
4. Try other breakpoints if you have time. Once done, all breakpoints can be cleared by Ctrl+Shift+<F9> or by choosing Clear All Breakpoints from the Debug menu. Individual breakpoints can be toggled using <F9> or the Breakpoint button on the toolbar.

Viewing Variables in the Locals Window:

The locals window shows the value of any variables within the scope of the current procedure. As execution switches from procedure to procedure, the contents of this window changes to reflect only the variables applicable to the current procedure. Repeat the above example and notice the values of X and Y also appear in the locals window.

Watch Expressions:

The Add Watch option on the Debug menu allows you to establish watch expressions for your application. Watch expressions can be variable values or logical expressions you want to view or test. Values of watch expressions are displayed in the watch window.

In break mode, you can use the Quick Watch button on the toolbar to add watch expressions you need. Simply put the cursor on the variable or expression you want to add to the watch list and click the Quick Watch button.

Watch expressions can be edited using the Edit Watch option on the Debug menu.

Watch Expression Example:

1. Set a breakpoint at the X = X + 1 line in the example.
2. Set a watch expression for the variable X. Run the application. Notice X appears in the watch window. Every time you re-start the application, the value of X changes.
3. At some point in the debug procedure, add a quick watch on Y. Notice it is now in the watch window.
4. Clear the breakpoint. Add a watch on the expression: X = Y. Set Watch Type to ‘Break When Value Is True.’ Run the application. Notice it goes into break mode and displays the watch window whenever X = Y. Delete this last watch expression.

Call Stack:

Selecting the Call Stack button from the toolbar (or pressing Ctrl+L or selecting Call Stack from the View menu) will display all active procedures that is those that have not been exited.

Call Stack helps you unravel situations with nested procedure calls to give you some idea of where you are in the application.

Call Stack Example:

1. Set a breakpoint on the Fcn = Cint() line in the general function procedure. Run the application. It will break at this line.

2. Press the Call Stack button. It will indicate you are currently in the Fcn procedure which was called from the Command1_Click procedure. Clear the breakpoint.

Single Stepping (Step Into):

While at a breakpoint, you may execute your program one line at a time by pressing <F8>, choosing the Step Into option in the Debug menu, or by clicking the Step Into button on the toolbar.

This process is single stepping. It allows you to watch how variables change (in the locals window) or how your form changes, one step at a time.

You may step through several lines at a time by using Run To Cursor option. With this option, click on a line below your current point of execution. Then press Ctrl+<F8> (or choose Run To Cursor in the Debug menu). The program will run through every line up to the cursor location, then stop.

Step Into Example:

1. Set a breakpoint on the Do line in the example. Run the application.
2. When the program breaks, use the Step Into button to single step through the program.
3. At some point, put the cursor on the Loop While line. Try the Run To Cursor option (press Ctrl+<F8>).

Procedure Stepping (Step Over):

While single stepping your program, if you come to a procedure call you know functions properly, you can perform procedure stepping. This simply executes the entire procedure at once, rather than one step at a time.

To move through a procedure in this manner, press Shift+<F8>, choose Step Over from the Debug menu, or press the Step Over button on the toolbar.

Step Over Example:

1. Run the previous example. Single step through it a couple of times.
2. One time through, when you are at the line calling the Fcn function, press the Step Over button. Notice how the program did not single step through the function as it did previously.

Function Exit (Step Out):

While stepping through your program, if you wish to complete the execution of a function you are in, without stepping through it line-by-line, choose the Step Out option. The function will be completed and you will be returned to the procedure accessing that function.

To perform this step out, press Ctrl+Shift+<F8>, choose Step Out from the Debug menu, or press the Step Out button on the toolbar.