

VBA Foundations, Part 9

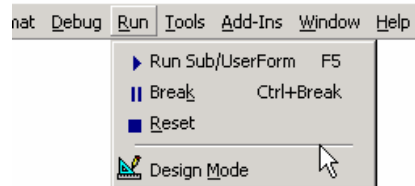
A Tutorial in VBA for Beginners—The Ninth of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

Welcome back for this month's exploration. Hopefully, you have been following along and trying your hand at creating your own VBA macros. In this episode I hope to explain the debugging process and provide a few helpful hints to keep your frustration level low and your creative juices flowing. But first, let's discuss the types of errors we might encounter which could require us to attempt to debug our code. There are basically three types of errors; compile errors, runtime errors, and logic errors. We have looked at how to include error trapping/handling in our routines in the last article which, hopefully, has gotten us to this point with fewer and fewer errors. The error handling techniques mentioned will generally cover the compile errors and the common runtime errors, so we will concentrate on the logic errors and the unexpected runtime errors which brings us to the debugging topic. For the most part your routines are now in working condition, but maybe they don't *always* give us the results we expect. In order to decipher and conquer these types of errors we must explore when and discover why they occur.

As you may have guessed, the process of debugging is usually more art than science; unfortunately, there is no precise series of steps that, when duplicated, always lead to the discovery of errors. So, it's a good thing that the VBA language and editor provides the tools we need to find both runtime and logic errors. It's an even better thing that it can also assist us in *fully* understanding what happens in our macro at every step while it executes. Of course, it goes without saying, the better you understand your application, the faster you can find the bugs. This is the heart of debugging. I trust that as you begin using the debugging tools you will explore in this episode, you will begin to understand your code inside and out. Let's explore the tools presented in groups as they are presented in the debug toolbar and in the menu pull downs. These tools are grouped with similar commands:

- The Operational Tools: These tools are available on the Standard Toolbar (next to the Undo and Redo buttons), the Debug Toolbar (See Figure 1), and under the "Run" Menu pull down (See Graphic to the right).
 - Run (F5) or "Continue" while in step through mode. This is the same as a full blown run. Not very useful for debugging unless you have previously placed break points in your code.
 - Break or Pause (Ctrl + Break) – Stops immediate execution of your code.
 - Reset or Stop – Stops execution of your code and clears objects out of memory.
- The Navigational Tools: The tools are available on the Debug Toolbar (See figure 1) and on the Debug Pulldown menu (See Figure 2).
 - Toggle BreakPoint (F9) – For use in advancing to a particular line of code.
 - Step Into (F8) – Begin the baby steps.
 - Step Over (Shift + F8) – Skip a line while stepping through your code. (Hint: dragging the yellow arrowhead, a.k.a. the *Call Stack Indicator*, located immediately to the left of the current line of code highlighted in yellow accomplishes the same thing. Pay attention to the change in your cursor as it moves in proximity to the yellow arrowhead. See Example above.)
 - Step Out (Ctrl + Shift + F8) – Complete running the current sub or function only



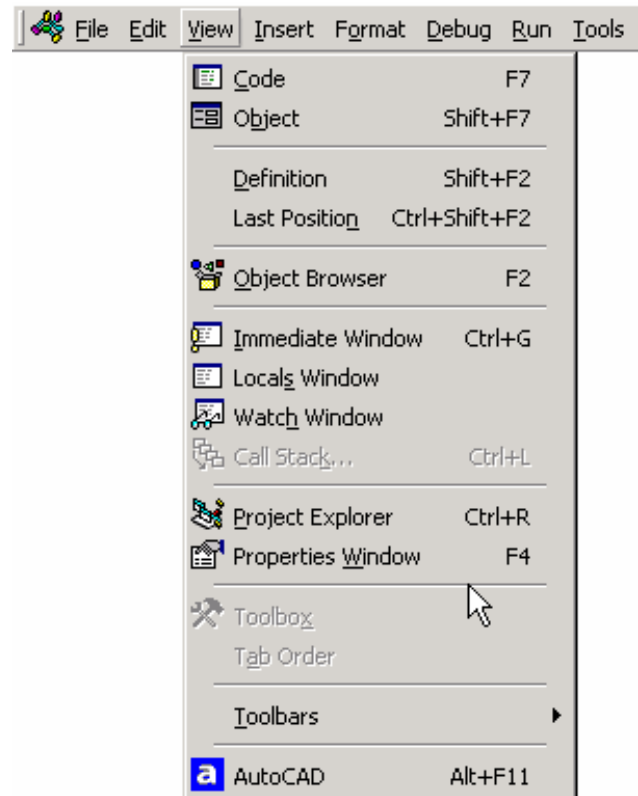
```
Sub ACADStartup()  
    valid = False  
    Update4ACAD_DVB = False  
    'localfile = "f:\cadd\acad.dvb"  
    'networkfile = "w:\thc-2i\vba\aca  
    Dim fso
```

VBA Foundations, Part 9

A Tutorial in VBA for Beginners—The Ninth of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

- Run to Cursor (Ctrl + F8) – Skip ahead to a line of code and click the left mouse button to mark the location, then hit the hot key combination (Ctrl + F8) to advance to your “marked” location.
- The Viewing or Status Tools. These tools can be found by exploring the “View” pull down menu (as shown to the right), or by selection from the “Debug” Toolbar (See Figure 1).
 - Locals Window – Usage: Checking values during execution. This window automatically displays all of the declared variables in the current procedure and their values. When the Locals window is visible, it is automatically updated every time there is a change from Run to Break mode or you navigate in the stack display. **Note:** *Global variables and variables in other projects are not accessible from the Locals window.*
 - Immediate Window (Ctrl + G) – Usage: Checking values and optional code during macro execution. This window allows you to: 1.) Type or paste a line of code and press ENTER to run it immediately. 2.) Copy and paste any code from this window into the Code window. **Note:** *This does not allow you to save code in the Immediate window, so if the code you are trying out works, then be sure to copy and paste it back into the code window before closing this window.*
 - Watch Window – Usage: Checking values during execution. A watch expression is an expression you define to be monitored in the Watch window. When your application enters break mode, the watch expressions you have selected previously appear in the Watch window where you can observe their values. This is useful for global variables which cannot be seen in the locals window, and **Note:** *You can drag selected expressions from the Code window into the Watch window while in break mode to see their values, but they must be in context or scope.* There are different types of watches as described below:
 - **Watch Type:** Determines how Visual Basic reacts to the expression.
 - **Watch Expression** — This type of watch displays the watch expression (variable, property, or Function) and its current value. When you enter break mode, the value is automatically updated.
 - **Break When Value Is True** — Execution automatically enters break mode when the expression evaluates to true or is any nonzero value **Note:** *This will not be a valid choice for string expressions.*



VBA Foundations, Part 9

A Tutorial in VBA for Beginners—The Ninth of a Twelve Part Series

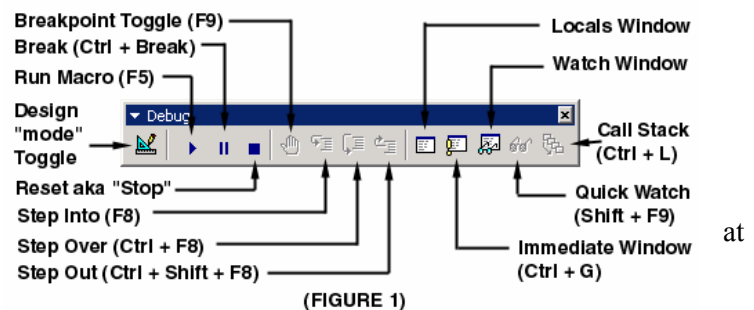
Richard L Binning / rbinning@attbi.com

- Break When Value Changes — Execution automatically enters break mode when the value of expression changes within the specified context (see below).
- Watch Context: Sets the scope of the variables watched in the expression.
 - Procedure — Displays the procedure name where the selected term resides (default is where it is first used or declared). Also available is the list of procedure(s) in which the expression is evaluated. You may select all procedures or a specific procedure context in which to evaluate the variable. (Hint: if you think the variable is being changed incorrectly, you can try to select available procedures to identify and watch the value as the macro progresses.)
 - Module — Displays the module name where the selected term resides. You may select all modules or a specific module context in which to evaluate the variable.
 - Project — Displays the name of the current project. Expressions can't be evaluated in a context outside of the current project.

Call Stack (Ctrl + L) –Displays the Call Stack dialog box, which lists the procedures that have started but are not completed. **Note:** *This command is only available while in break mode.* This really should be part of the navigation tools since it points you to the subroutines or functions that call or follow the current code line, but it is available from the locals window through choice of a button, by selection from the “View” menu pulldown as shown in the graphic above, or by using the Hot Key combination (Ctrl + L).

- How it works: *When executing code in a sub or procedure or function, that procedure is added to a list of active “calls”. Each time a procedure calls another procedure, it is added to the list. Called procedures are removed from the list when execution returns to the calling procedure. **Note:** Procedures called from the Immediate window are also added to the calls list.*

Before we get started, lets take a look at the tools provided for debugging. (See Figure 1) This graphic shows the available commands and tools, you the programmer, will use when debugging code in the vba editor. I want you to take special note of the “Step Into” tool and the “Immediate Window” tool. The “Step Into” tool will allow you to begin running your code a line a time. Try it now. Open the VBA editor and using the vbamanager command in AutoCAD



(FIGURE 1)

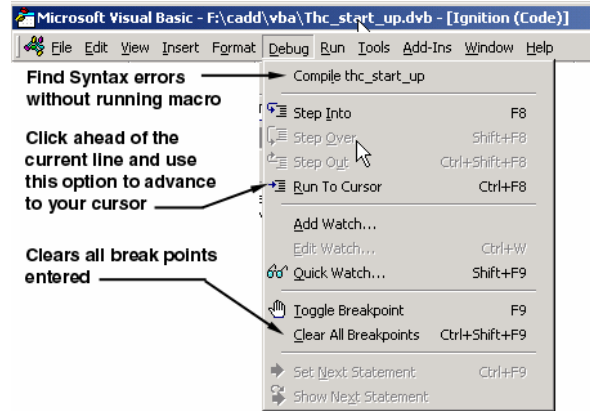
load your current macro or one of the examples installed with AutoCAD and hit the “F8” key. Nothing happens? Make sure you find a subroutine in the code window and place your cursor in the routine by left clicking the mouse somewhere within the subroutine. Now hit that “F8” key. Did the first line, “Private Sub Something()” suddenly become highlighted in bright yellow. Do you see the yellow arrowhead to the far left of that first line? For extra credit you can drag that arrowhead by clicking and dragging it, but only to lines in your code that have some activity assigned to them (It will not work for declaration lines. Hint: if the line begins with Dim then you can’t park the arrowhead there.) What does dragging the arrowhead around accomplish? How might we make use of this ability? Did you notice as you advanced through your code hitting the “F8” that the “yellow” or current line never stopped on declaration statements?

VBA Foundations, Part 9

A Tutorial in VBA for Beginners—The Ninth of a Twelve Part Series

Richard L. Binning / rbinning@attbi.com

You should already be familiar with the “Run Macro” button if you’ve been trying out any of the code or examples provided. You may notice that some of the tools are “grayed out” as shown in the graphics in this article, please be assured that they are available when needed, which is usually deep in debug mode when you are stepping through your code. The “F9” key will create a breakpoint in your code at the line you have placed the cursor prior to hitting it. You can also click in the “gutter” immediately to the left of a line of code to place a breakpoint. Take a look at the code window and see if you can find this “gutter”. It is the same place that displays the “Call Stack Indicator” mentioned above. Placing a breakpoint in your code allows you to run your code as in normal execution until it gets to the line containing a breakpoint. The code or macro will then stop executing at this breakpoint and immediately display the code window with your code highlighted at that line and the “Call Stack Indicator” displayed prominently in the gutter to the left of this line and visually on “top” of the breakpoint symbol. This action can often save you from stepping through (“F8”) every line of your code one line at time. This is especially convenient to use if a particular line of code is erroring out and throwing you out of AutoCAD. You can try to debug this type of problem by placing a breakpoint immediately prior to the offending code provided you know where this error is being encountered.



(FIGURE 2)

Now take a look at the menu as shown in Figure 2. Note that some tools are only available via hot key or menu pull down, since they are not on the toolbar shown above in Figure 1. Of particular interest are the tools I have attached comments to in Figure 2. Note: you can compile your macro to get the editor to find syntax errors without actually running your macro. Also, you will want to become familiar with the “Run to Cursor” tool as you are stepping through your code.

As always, use the on-line help to further explain these concepts as necessary. If you are really stumped, please send in your questions to the VBA guild or the email address at the top of this article. See you on the guilds or in the next issue of AUGIWorld™ magazine coming soon to a mailbox near you. Next issue we will look at the Object Model of AutoCAD and begin putting some more extensive code together.