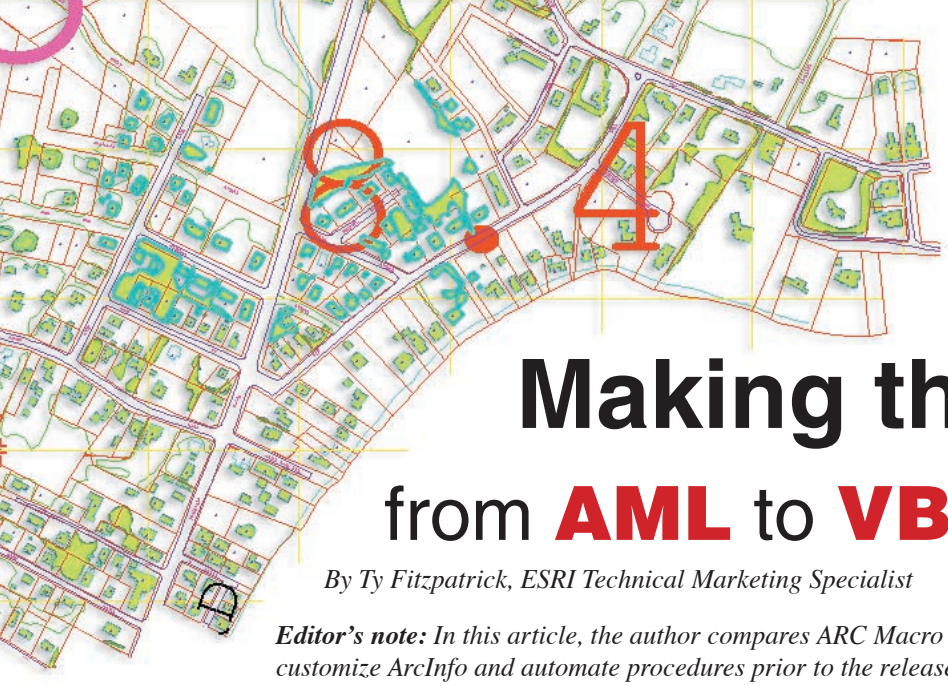
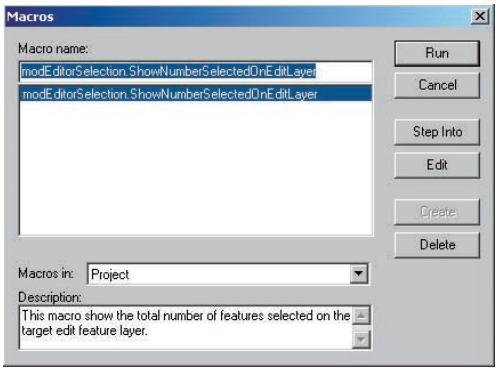


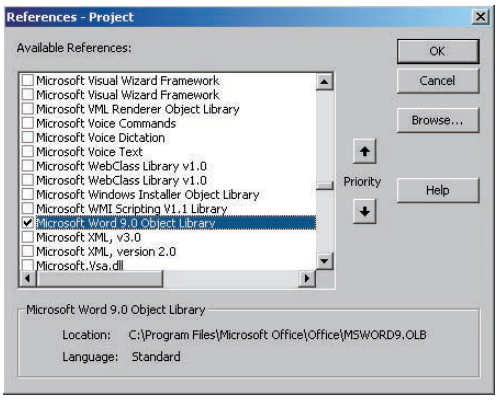
Making the Switch from AML to VBA

By Ty Fitzpatrick, ESRI Technical Marketing Specialist

Editor's note: *In this article, the author compares ARC Macro Language (AML), the language used to customize ArcInfo and automate procedures prior to the release of ArcGIS, with Microsoft's Visual Basic for Applications (VBA), the built-in development environment for ArcGIS. He takes a simple application for returning the number of selected features written in AML and shows how to quickly write the same application in VBA and provide additional functionality with very little effort.*



In ArcGIS, commands can be written explicitly and saved as a macro. Choose Tools > Macros to bring up the Macro management tool for creating, running, and deleting macros.



In addition to the classes available through the default set of class libraries in ArcGIS, additional libraries from other applications, such as Microsoft Word, can be added to a project to provide access to more objects.

The built-in development environment for ArcGIS is Microsoft's VBA. With VBA, GIS users can rapidly create macros and user interfaces for ArcMap, ArcCatalog, and ArcScene. Because these applications are COM-based, any COM-compliant language can be used to develop applications using ArcObjects, ESRI's component object model. However, the quickest and easiest way to extend the capabilities of ArcGIS is to use VBA. For GIS users familiar with AML, VBA provides more robust debugging tools and access to the controls and libraries available through Windows and COM-based applications. Customizations can easily be saved and shared within an organization.

In the Beginning…There Was AML

AML provides programming capabilities and a set of tools for tailoring the user interface of an ArcInfo Workstation application. Modeled on Prime Computer's Command Procedural Language (CPL), AML was created by ESRI to unify application programming in ArcInfo across all platforms. It is a mature development environment that supports functions, variables, and basic programming constructions such as branching. However, it lacks an integrated development environment with common controls and access to the Windows application program interface (API), making the development of applications that are integrated with other desktop programs more challenging.

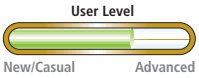
Why Use VBA?

Object-oriented programming has distinct advantages over procedural programming. These advantages stem from three characteristics that are present in VBA—encapsulation, polymorphism, and inherited functionality. These characteristics help developers create smaller, more manageable code bases.

- Encapsulation hides the data and implementation in an object. Developers only deal with the exposed properties and methods of an object.
- Polymorphism lets a single object perform multiple tasks, simplifies code management, and reduces redundancy. Code redundancy can be a problem when using procedural languages such as AML.
- Although inheritance in the classic sense is not implemented in VBA, VBA objects can obtain functionality from other objects using the Implements statement.

VBA provides the same tools as Visual Basic (VB) but provides them in an existing application—in this case the core ArcGIS applications, ArcMap and ArcCatalog, and the ArcGIS extension, ArcScene. VBA is built into each of these applications and is immediately available from the menu by choosing Tools > Macros. It is the easiest method for implementing customization that requires automation or functionality beyond existing commands in ArcGIS. However, code cannot be compiled in VBA, and all customizations must be saved to a map document or template. A stand-alone programming environment such as VB (or Visual C++ or Delphi) is required to create full-blown custom applications and controls.

VBA supplies many new controls in the Windows environment such as TreeViews, Grids, and CommonDialogs. In addition to the classes (i.e., blueprints for objects) available through the default set of class libraries, additional libraries from other applications, such as Internet Explorer, Crystal Reports, and Visio, can be added to a project to provide access to more objects.



AML

Figure 1
(in ArcEdit)

```
&if [show program] nc 'ARCEDIT' &then &ret ARCEDIT TOOL:
You are not in Arcedit.

&if [null [show edit]] &then &ret No edit cover set.

&if [show ef] cn 'NONE' &then &ret No edit feature set.

&s oldmessage [show &messages]
&message &popup
&type [show number select]Features(s) Selected.
&message %oldmessage%

&return
```

AML

Figure 2
(in ArcPlot)

```
&if [show program] nc 'ARCPLOT' &then &ret ARCPLOT TOOL: You
are not in Arcplot.

&s flist listofapsel.txt

&s oldmessage [show &messages]
&message &off
&watch %flist%
listselect
&watch &off
&message %oldmessage%

&s fileunit [open %flist% fopen -read]

&if %fopen% <> 0 &then
 &ret ~ Unable to open file: %flist%

&s readstat = 0
&s i = 1
&do &until %readstat% > 0
 &s record%i% [read %fileunit% readstat]
 &if %readstat% = 0 &then &s i = [calc %i% + 1]
&end
&dv record%i%
&s i = [calc %i% - 1]

&if [close %fileunit%] <> 0 &then
 &type Could not close file: %flist%

&if [delete %flist% -file] <> 0 &then
 &type Could not delete file: %flist%

&s oldmessage [show &messages]
&message &popup
&lv &format '%1,1%:%2%' record*
&message %oldmessage%

&return
```

Flexibility in declaring functions and variables is important for managing data storage and interfaces. Hundreds of common Windows API functions are available to the VBA programmer. Variables are also more robust in both scope and type in VBA. Declaring variables in AML is restricted to the &Setvar directive at the local, program, and global scope. This is similar to the scope of variables in VBA, but declarations at the module level in VBA can be either public or private. Variables in AML can be character strings, integer numbers, real numbers, or Boolean data types. VBA has these types and adds double, float, variant, arrays, collections, and object data types, allowing for more efficient memory allocation.

Using VBA also provides a level of customization completely beyond what is available in AML. VBA lets developers access the technology framework of ArcGIS—ArcObjects. With these components, ArcGIS can be programmatically enhanced with new tools, improved work flow, and, through extending the ArcGIS data model, new custom feature types.

Finally, because VBA is part of the mainstream IT world, many people are conversant with it. Finding a developer to work on projects is much easier. Although existing AML programmers who want to take advantage of VBA will have to learn object-oriented programming, the benefits of a robust programming environment, easier code maintenance, and more available resources will more than repay the effort expended.

Making Tasks Easier

Although both AML and VBA let the user automate processes and create interfaces, the advantages of using VBA can be seen in terms of level of control and functionality, ease of creation, and choices for implementing and distributing customizations.

AML programs for automating processes can be created by capturing commands that have been entered during an ArcInfo session using the &WATCH directive. Applying &CWTA converts the watch file to an AML file by removing system output. Since AML programs, menus, and menu programs are written in ASCII, they can also be created by directly typing in a text editor.

Continued on page 38

Making the Switch from AML to VBA

Continued from page 37

In ArcGIS, commands can be written explicitly and saved as a macro. Choose Tools > Macros to bring up the Macro management tool for creating, running, and deleting macros. Alternatively, choose Tools > Macros >Visual Basic Editor to type a macro into VB Editor. VBA speeds code writing by stubbing out the base source code for the macro in new module, supplying tips on syntax (i.e., Auto Quick Info), and automatically completing code and variable names that VBA recognizes. These features also help avoid bugs caused by simple typing errors and eliminates the need for an outside source for syntax information.

In AML, simple menus that let users make selections from a list of coverages, files, or other items can be created using the [GET…] function. However, even with Form menus, the most robust AML menu type, the selection of control types (i.e., the sliders, checklists, and other widgets in AML) is limited. With AML, the programmer also has limited control over appearance, size, and location of interfaces.

For AML developers who have used FormEdit to create form menus, creating interfaces in VBA will seem familar. Controls are dragged, placed, and sized on a VBA form, and properties for controls are set in a manner similar to one used in designing a Form menu. In addition to incorporating more types of controls for the user to interact with, in VBA, actions can be triggered by events such as a form loading or a selection change.

Getting the Bugs Out

Of the eight debugging and monitoring tools available in AML, &SEVERITY, &WATCH, &ECHO, &TEST, and &TYPE are the most useful directives. To generate error messages, the program must be executed in some fashion. This makes isolating errors in very large code bases, which often contain dozens of variables, challenging.

In VBA, all code for forms, modules, and classes is handled in one programming environment using the same debugging tools. Debugging a VBA macro is easy because it runs in the same process space as ArcMap. In addition to the Autocompletion and Auto Quick Info features that avoid errors in the first place, VBA includes tools, such as watch expressions, that help a programmer observe the behavior of variables and statements. Clicking on a line of code sets a break point that will stop program execution at that point at run time. The code can then be analyzed line by line to locate errors.

VBA

Figure 3
(create interface to execute the macro)

```
Sub DisplayForm()
  frmShowSelected.Show vbModal
End Sub

Private Sub cmdShowNumSelected_Click()
  Call ShowNumberSelected
End Sub

Figure 4
[execute macro based on an event]

Option Explicit

Private WithEvents w_pEditor As Editor

Private Sub Class_Initialize()

  Dim pUid As UID
  Dim pEditor As IEditor

  Set pUid = New UID
  pUid.Value = "esriCore.Editor"
  Set pEditor = Application.FindExtensionByCLSID(pUid)
  Set w_pEditor = pEditor

End Sub

Private Sub w_pEditor_OnSelectionChanged()
  Call ShowNumberSelected
End Sub
```

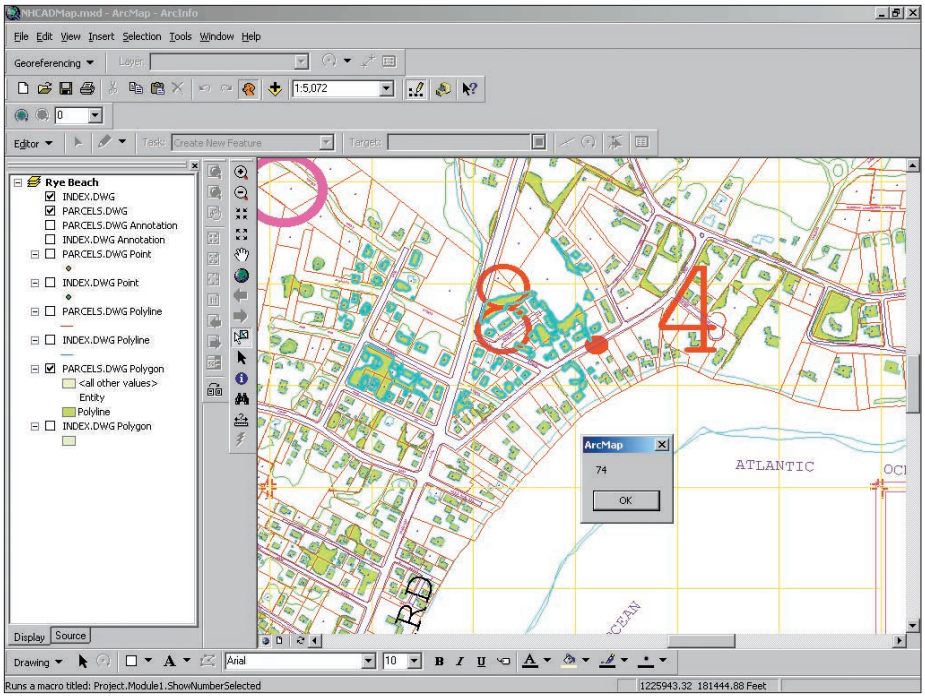
Many Ways to Run a Macro

A sample macro written in AML (Figures 1 and 2) and VBA (Figure 3) displays the number of selected features. The AML macro can run from the command line or can be attached to a widget in the Form menu.

The VBA macro can be run from the Macro management tool, attached to a button on the interface, or triggered by an event. For customizations that will be used by others, running macros from the Macro management tool is not the best choice. Adding a button to the interface and attaching the macro is a better solution.

1. To add a button, right-click on any displayed toolbar in the interface. In the context menu, select Customize.
2. In the Customize dialog box, click on the Commands tab. Select Macros from the pick list in the left pane.
3. In the right pane of the dialog box, select the new macro and drag it onto an existing toolbar. A new command button that will execute the code is created.

The next step is to create a user interface that can be used to run the macro for returning the number of selected features. The code in Figure 3 displays a form with a single button that calls the ShowNumberSelected macro when clicked. As a final enhancement, the code in Figure 4 triggers the ShowNumberSelected macro when the selection is changed (i.e., by an event) using the OnSelectionChanged method from the Editor extension. Events are messages sent from the server to the client application through public properties on an object. This example illustrates the flexibility and usability of VBA and may suggest some new ways for developing applications.



The code in Figure 3 generates a message box that shows the number of selected features.

Distributing Customizations and Managing Code

Customizations made using VBA can be stored in the Normal template or a single map document. The Normal template of each application for each user is stored in the Windows Profile directory. These customizations will be available to all documents of that type owned by that user. Code saved to individual map documents (only valid with ArcMap and ArcScene) is accessible only from that individual document. Making changes to a map document and testing those changes is an excellent way to prototype code without disturbing the code base stored in the Normal template.

Applications can be password protected so only developers have access to the code base. Placing a modified Normal template in \Arcexe81\bin\templates directory is an easy way to distribute customizations to GIS users. When a user without a Normal template for that application starts the application, the modified Normal template will be copied to the user's Profile directory.

Modifying the Normal template is a very useful method for distributing customizations within organizations that require a limited number of applications. For organizations managing a larger code base stored in a central repository where the "check-in check-out" method is used, VB or Visual C++ provides a better solution for writing applications. Even when applications will be implemented in VB, VBA is excellent for prototyping new ideas and testing code.

Conclusion

While AML is still a useful tool, AML programmers should consider taking advantage of the efficiency and flexibility of VBA for extending functionality in ArcGIS applications. With VBA, programmers can use ArcObjects to develop integrated GIS applications.

In addition to the online help that comes with the Visual Basic Editor in ArcGIS, there are many resources available for learning VBA. Both instructor-led and Virtual Campus courses on using VBA with ESRI software are available. Visit the ESRI Web site for information on current course offerings. The ArcObjects Developer Web site (**arconline.esri.com/arcobjectsonline**) also provides specific information on programming with ArcObjects as does the two-volume reference *Exploring ArcObjects,* edited by Michael Zeiler and available from the GIS Store (**www.esri.com/gisstore**). AU

About the Author

Ty Fitzpatrick received a degree in geography from the University of Delaware. While attending college, he worked for the Water Resources Agency of New Castle County Delaware. He began working for ESRI seven years ago as a technical support technician and has been in technical industry marketing for the past five years. He creates demos and provides support for tradeshows and benchmarks.

Fast, Accurate, and **Lazy** in VBA

The integrated development environment that comes with ArcGIS, Visual Basic for Applications (VBA), rewards you for taking the easy way out. Programmers moving from a procedural language, such as ARC Macro Language (AML), may be pleasantly surprised how quickly they can become productive in VBA by taking advantage of the timely help provided by several utilities that are built into the Visual Basic Editor.

Type an object variable that the Visual Basic Editor recognizes followed by a period, and it will list all the methods and properties available for that object. To select a method or property, either type the first few letters of its name or scroll through the list and choose the one to use. Press Tab, Enter, or Space and the highlighted method or property is added to the code. By letting VBA automatically complete the names of methods and properties, you can avoid run-time errors caused by simple typographical errors. If the code completion list does not appear, it means that the Visual Basic Editor did not recognize the object variable, and you probably have an error earlier in your code—something that's also good to know.

Auto Quick Info, another feature, can flatten out the learning curve for new users. As you type a statement that the Visual Basic Editor recognizes, a tool tip-like box pops up displaying the proper syntax, all the arguments—both required and optional—and the return data types if the statement returns a value. No need to flip through a reference book, open another window, or scratch your head trying to recall any of these details.

Avoid aggravation and save time when debugging. Use the automatic completion of variable names feature to eliminate, or at least minimize, errors caused by mistyped variable names.