From "Digital Mapping Techniques '10—Workshop Proceedings"
U.S. Geological Survey Open-File Report 2012–1171
http://pubs.usgs.gov/of/2012/1171/

Automation in ArcGIS 10: Understanding Changes in Methods of Customization and Options for Migration of Legacy Code

By Andrew L. Wunderlich

Tectonics & Structural Geology Research Group
Department of Earth and Planetary Sciences
and Science Alliance Center of Excellence
306 Earth & Planetary Sciences Building
1412 Circle Drive
University of Tennessee
Knoxville, TN 37996-1410
Telephone: (865) 974-6448
Fax: (865) 974-9326
email: *gibbon@utk.edu*

Abstract

Important changes to how customizations and automations are created have been included in the latest update to Esri ArcGIS software, version 10. Microsoft has dropped support for Component Object Model (COM)-based programming languages Visual Basic 6 and Visual Basic for Applications (VBA) and is emphasizing a shift to Java- and .NET-compliant languages. As a result of this change, Esri is following suit by removing the familiar VBA development environment from their products, discontinuing its support, and promoting new scripting and application development alternatives. This paper seeks to describe the process of making the change from COM to .NET by (1) clarifying the reasons for the change, (2) discussing the leading vendor-supported, alternative scripting methods, (3) explaining the new Add-in model for customizing the ArcGIS interface, and (4) describing the most common and important differences between VBA and VB.NET code that are encountered during a conversion from previous versions of ArcGIS to version 10.

Introduction

With the release of ArcGIS 10, Esri has implemented many new features and updated components to their popular GIS software package. One of the most significant changes for power users is the way in which scripting, automations, and customizations to the user interface are handled. At the DMT'09 meeting in Morgantown, W.Va., I gave a presentation entitled "Improving ArcGIS workflow: Automation using Visual Basic for Applications (VBA)" in which I described using VBA to customize the ArcMap 9.x interface (Wunderlich, 2010). Since that time, the automations I described and posted to the Esri ArcScripts Web site have been downloaded over 1,600 times collectively, and I have received many emails regarding that presentation and the scripts described within it. Clearly, there is still a great interest in VBA applications, but as support for these applications is waning, it is now necessary to update these applications to work in the new customization framework of ArcGIS.

Since the debut of the ArcGIS suite (v. 8.0) more than 10 years ago, Visual Basic for Applications (VBA) has been its supported, integrated scripting language. In ArcGIS 10, support for the Microsoft Component Object Model (COM)-based VBA has been dropped, and a shift to Java- and .NET Framework-compliant programming languages is being emphasized. Customizations to the ArcGIS interface now require new methods of developing and debugging applications and scripts. Scripts, tools, and user interfaces developed using VBA will have to be converted to a compliant scripting language (such as Python) or converted to Java or .NET.

Customizing ArcGIS 10

One of the great advantages of the ArcGIS framework is that it is open for users to create customizations at any level of expertise, across the entire spectrum of the software's functionality. The most common forms of customization in ArcGIS 10 remain fundamentally the same as they were at 9.x, but with some notable differences, as explained below:

- Layer files, styles, representations, and templates in ArcMap documents
- Model Builder for creating geoprocessing workflows
- Python scripting (now with ArcPy) for advanced geoprocessing and map production
- Custom buttons and user interfaces (Add-ins) are created with Java, VB.NET, or C# *outside* ArcGIS using Microsoft Visual Studio (or Eclipse).

The first two items in this list remain relatively unchanged from 9.x. The more significant changes are the increased support for Python scripting and the deprecation of VBA and adoption of the Add-in component model, which uses the .NET development environment. The remainder of the discussion in this paper regarding customizations will focus on the latter two forms of customization.

Python Scripting and the New ArcPy Site-Package

With the launch of ArcGIS 10, Esri has fully embraced Python as its scripting language of choice for geoprocessing and automation of map production. Python is an open-source, cross-platform scripting language that has been in extensive use since the early 2000s. Some of the advantages of Python include its gentle learning curve, highly readable code structure, and runtime interpretation (no compilation or system registration is necessary). The ability to use Python for scripting has existed within the ArcGIS framework since version 9.0, mainly for creating geoprocessing scripts for use within ArcToolbox. Until now, Python was rather limited in functionality because many components of the ArcGIS framework were not exposed to Python. To improve the functionality of Python, Esri created the ArcPy site-package and added a command-line Python scripting window to all ArcGIS applications in order to allow scripts to be loaded and run on-the-fly within the individual applications (for example, in ArcMap and ArcCatalog). The ArcGIS Help describes ArcPy as an add-on to Python that "…provides access to geoprocessing tools as well as additional functions, classes, and modules that allow you to create simple or complex workflows quickly and easily" (ArcGIS Resource Center, 2011a).

Esri states that ArcPy has five major organizational groups: tools, environments, functions, classes, and modules (ArcGIS Resource Center, 2011a). ArcPy tools expose all available Toolbox tools, depending on your license level. This includes basic tools such as Buffer, Copy, Append, and Dissolve, and additional tools that are exposed by ArcEditor or ArcInfo license levels (for example, Densify, Snap), plus any tools exposed by licensed extensions such as the Hillshade tool in Spatial Analyst. Environments allow you to modify the tool's parameters that are used while executing, including: snapping tolerance, cell size for raster analysis, and input and output workspaces. Functions are general-use with no license dependence. They are used to do basic things such as checking for the existence of an object, querying feature class parameters such as the spatial reference, and refreshing the map view. Classes are "helpers" that aid the creation of objects (also known as instances) such as a spatial reference, a coordinate pair (point), or a cursor to store a set of features to be processed iteratively. These "instances" of objects can be referenced as often as needed during the execution of a script. Modules are groups of classes, used for referencing a specific set of functions related to a particular aspect of ArcGIS. For example, the Mapping module gives the user access to functions that open map documents, manipulate layers, and export or print maps. For more information about using Python and ArcPy in ArcGIS, see the ArcGIS Desktop Help topic "*What is ArcPy?*"

Using the Python window and utilizing the functions exposed by ArcPy, one can create some very powerful automations to aid in speeding up repetitive geoprocessing and map creation tasks. Consider the following example of a workflow that could be scripted with Python:

> An organization making an atlas is trying to create a graphical index of the atlas pages that shows the extent of each larger scale regional map on a small-scale map of the world. The process to do this manually would go something like this:
>
> - Open the ArcMap document of the atlas page.
> - Create a feature class to store a polygon that represents the spatial extent of the page.
> - Query the extent of the map and draw the corresponding polygon.
> - Create fields and calculate values in the polygon feature class attribute table that identify the map name and map scale.
> - Close the atlas page map document.
> - Open the map document that represents the graphical index.
> - Add the polygon feature class created in the previous steps.
> - Set the layer properties to label the polygon with the map name.
> - Save and close the graphical index map document.

Each step in this scenario can be accomplished by accessing various ArcPy modules and their classes: open and close map documents, create feature classes, query the map extent, create a feature based on the extent, add fields to the attribute table, calculate values based on map parameters queried from the document, add layers to map documents, and set map layer-labeling parameters. The user could additionally make the process iterative, whereby the script opens each map page, carries out the process of creating the extent polygon, and adds it to the graphical index map document in turn. If you or your organization has handled this type of process with a VBA script in the past, then a Python script will probably work very well for you in ArcGIS 10. Automation of repetitive processes that do not require a lot of user interaction are prime candidates for Python scripting.

Limitations of Python

The example presented in the previous section is one that is well suited to a Python solution. Many other tasks, such as generating empty databases from a template, validating database structures, and automated export of maps, are all perfect candidates for Python scripting solutions. However, as a development environment, Python has two major shortcomings when compared to more robust development environments such as Microsoft Visual Studio .NET (or even VBA). First, not all components of ArcGIS are exposed to Python. Geoprocessing tools and many functions for creating and interrogating datasets and map documents are available, but customizations to tool functions, or building new tools, is really not possible because access to the full library of ArcObjects is not available. Second, the editing and debugging capabilities of Python are limited. Third-party editors for Python, such as PythonWin (which is included with the ArcGIS 10 application suite), improve the readability and editability of the code, but do not have the power to debug and validate code as Visual Studio can. These are minor shortcomings and should not prevent users from developing scripts with Python. It is, after all, a *scripting* language, not an application development environment.

This brings us to the discussion of where the usefulness of Python gives way to more robust programming solutions. Python is not unlimited in capability and it is not appropriate for some important types of customizations that were accomplished with VBA in the past. Python's *most* important limitations compared to VBA or .NET are (1) it cannot listen for or respond to events within the ArcGIS application framework and (2) you cannot create any custom user interfaces that are tied directly to the application framework, such as buttons, toolbars, or user forms. If you have custom buttons, toolbars, combo boxes, editor extensions, or interactive forms that you need to function in ArcGIS 10, they will have to be recreated in the new system of customizations that Esri has implemented; these are called Add-ins.

Goodbye, VBA! Introducing Add-ins for ArcGIS

The Add-in model for customizing ArcGIS is a new feature, added at version 10. Add-ins have taken the place of VBA as the method for extending the user interface of Microsoft Windows-based applications that support customization. According to the ArcGIS Resource Center, the Add-in model "provides you with a declaratively-based framework for creating a collection of customizations conveniently packaged within a single compressed file" (ArcGIS Resource Center, 2011b). A more detailed discussion of the components of an Add-in for ArcGIS is presented in the next section. As for VBA, Esri has decided to continue support in a *very* limited fashion to facilitate the changeover. By default, VBA is not installed with ArcGIS 10, but it can be installed separately. If you choose to install VBA, a special license keycode must be requested from Esri to make it work, as if it were an extension such as Spatial Analyst. Esri is strongly discouraging any development using VBA and suggests that users migrate code to a supported language such as VB.NET, C#, C++, or Python. After version 10.0, VBA will be completely removed from ArcGIS and users will need to use the Add-in model for customizations to the user interface.

Overview of Add-ins

Add-ins are written in either a .NET or Java development environment. Major development packages that are supported by Esri include Eclipse and Microsoft Visual Studio (MSVS) 2008 and 2010, including the MSVS 2008 Express edition, which can be downloaded free of charge *from Microsoft*. [Note: As of this writing, only the *2008* version of MSVS *Express* is supported by the ArcObjects software development kit (SDK) for the Microsoft .NET Framework. See the SDK *system requirements* page for more information.] The ArcObjects SDK includes a wizard that integrates into these development environments to easily build new projects. The wizard handles the creation of all the required components of the Add-in. These components consist of the .NET or Java class (the code) and an XML file that describes the Add-in to ArcGIS, as well as any icons or picture resources required by the Add-in. When a new Add-in is created with the wizard, the user can name it, describe it, and specify its type. Then the required components are created and opened in the Solution Explorer of MSVS for the user to add the custom code.

There are many types of Add-ins available for ArcGIS when using the wizard:

- Buttons
- Tools
- Combo boxes
- Menus and context menus

- Multi-items
- Toolbars
- Tool palettes
- Dockable windows
- Application extensions
- Editor extensions

For more information about each type, and help in choosing the right one for your needs, see the article "*Building add-ins for ArcGIS Desktop*" in the ArcGIS Resource Center. The most common customizations are buttons, tools, combo boxes, toolbars, and editor extensions. Also, customizations that display information or accept input from the operator may present the user with a Windows user form, which can easily be added to button and tool projects.

Once an Add-in is created and custom code is written or converted from an existing VBA or VB6 project, it can be built and registered for use in ArcGIS. One of the advantages to the new system of Add-ins is that the user no longer has to paste code or load forms into the VBA editor manually or run an installation program to enable the Add-in to be recognized by ArcGIS. Simply building the project in MSVS will automatically register the Add-in on the computer used to develop it, and the .ESRIAddin file created during that process can be distributed to others and registered for use in ArcGIS simply by double-clicking the file and following the prompts.

Common Issues Encountered when Migrating Code

The purpose of this paper is not to step through a conversion of code from VBA to .NET, as there are many articles and resources available on the Web to help with specific details about the process. It is also difficult in a concise paper to explain *all* the nuances of conversion, so I have compiled some helpful resources for conversion, as well as links to articles that describe specific conversion tasks. Instead, I want to emphasize major differences between developing in .NET (specifically Visual Basic .NET) and in the VBA environment, including some of the most common errors you will encounter and some of the key differences in properties and syntax.

One consideration when beginning the conversion is the software environment in which you will be redeveloping your VBA projects. If you choose to use the freely available MSVS Express (VB.NET or C#), keep in mind that only the 2008 version is currently compatible with the ArcObjects SDK. The new project wizard will not be available if you use any 2010 Express edition (the wizard is available in all full versions of MSVS, 2008 and 2010). Also, one of the most helpful components of the SDK is the ArcGIS Snippet Finder, which is only available with full versions of MSVS and will not be available in any Express edition. Snippet Finder allows you to search for bits of code already written in .NET that can be inserted into your project.

Importing VBA Code to an Add-in

The first step for most code conversions is simply to import or copy/paste code from a VBA project to a new add-in project, in Visual Studio. In ArcGIS 9.x and earlier, code is commonly stored in the ThisDocument class of the "Normal" template of the application being customized (for example, in ArcMap or ArcCatalog). The code in that class can be exported to a file, typically called ThisDocument.cls. When you begin a new project in Visual Studio, you can import the contents of ThisDocument.cls to a new class in your project. Then you can take advantage of some of the error correction features of Visual Studio, which are far more advanced than the VBA editor's debugging capabilities. Depending on how many custom buttons and functions are stored in the ThisDocument class, you will probably need to split the code among several new classes and (or) several add-ins.

Once the code you need is imported into your project, notice the zigzag underlines on certain parts of the code. These denote errors in the code. The MSVS Error List inventories all the errors in the code, and by stepping through each error and either making the suggested correction or reworking the syntax, errors can be quickly identified and corrected. If you find that some of the errors are repetitive (as some of the errors undoubtedly will be) you can use the "Find All References" command from the context menu that pops up when right-clicking on an object in the code window. This command returns, to the Find Symbol Results window, all instances of a particular property or function within your project and gives you important information such as the object definition and line and character number of the occurrence (fig. 1). This is very helpful when you need to edit all instances of a reference or property *consistently*. Hovering the cursor over an error brings up a small exclamation point with a dropdown arrow that gives suggestions to correct errors (fig. 2), but be careful, the suggestions are based on the currently loaded references and might not always have the correct solution!

Another invaluable tool when working through a code conversion is the MSVS Object Browser. Once the ArcObjects SDK is installed, the Object Browser has access to every class object, interface, and property or method that can be accessed programmatically in your customization. This is especially helpful when an error description in the code informs you that a particular method is not associated with the object that it is referencing, or that a previously recognized class type is not defined and needs an object reference. By using the Object Browser's search function, you can search for the method and see which objects support it; this generally helps you find the correct object reference and fix the error (fig. 3).

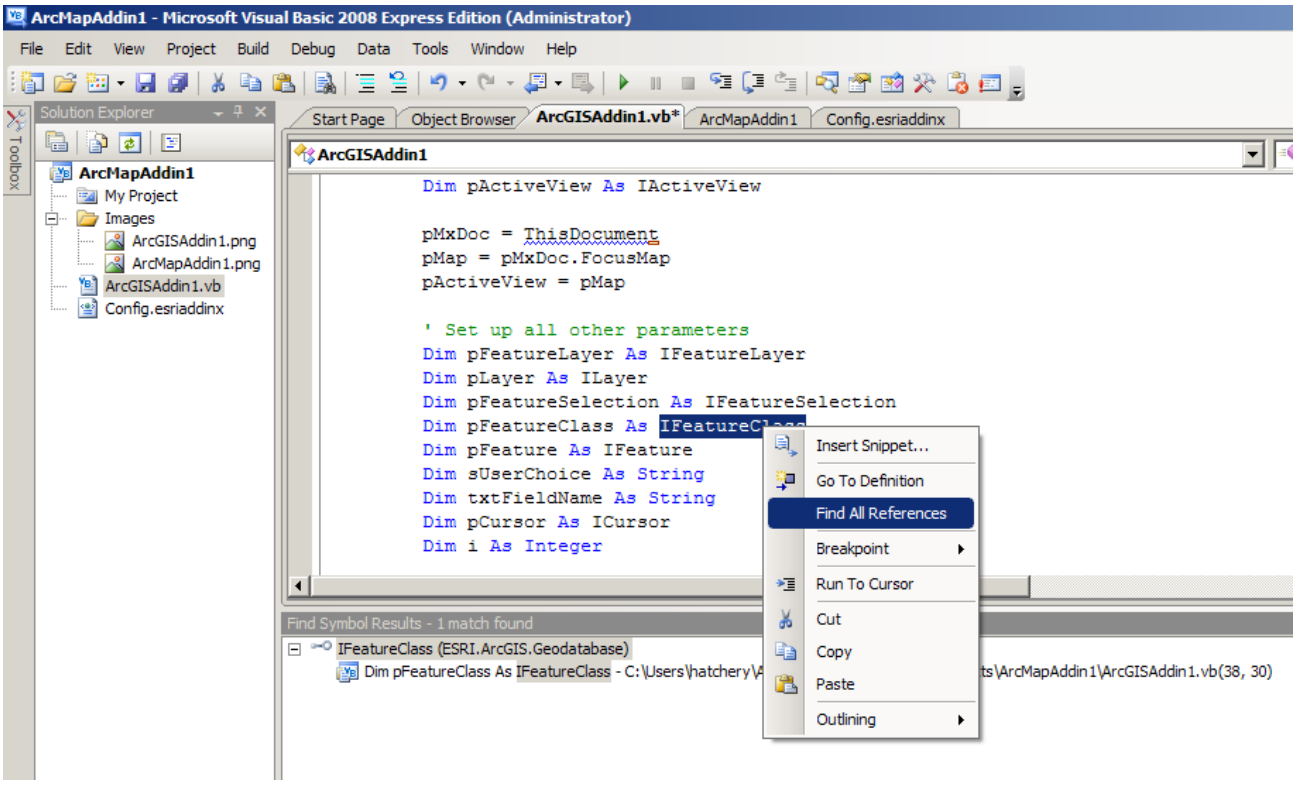

Figure 1. Use the "Find All References" command to identify all instances of a particular code object.

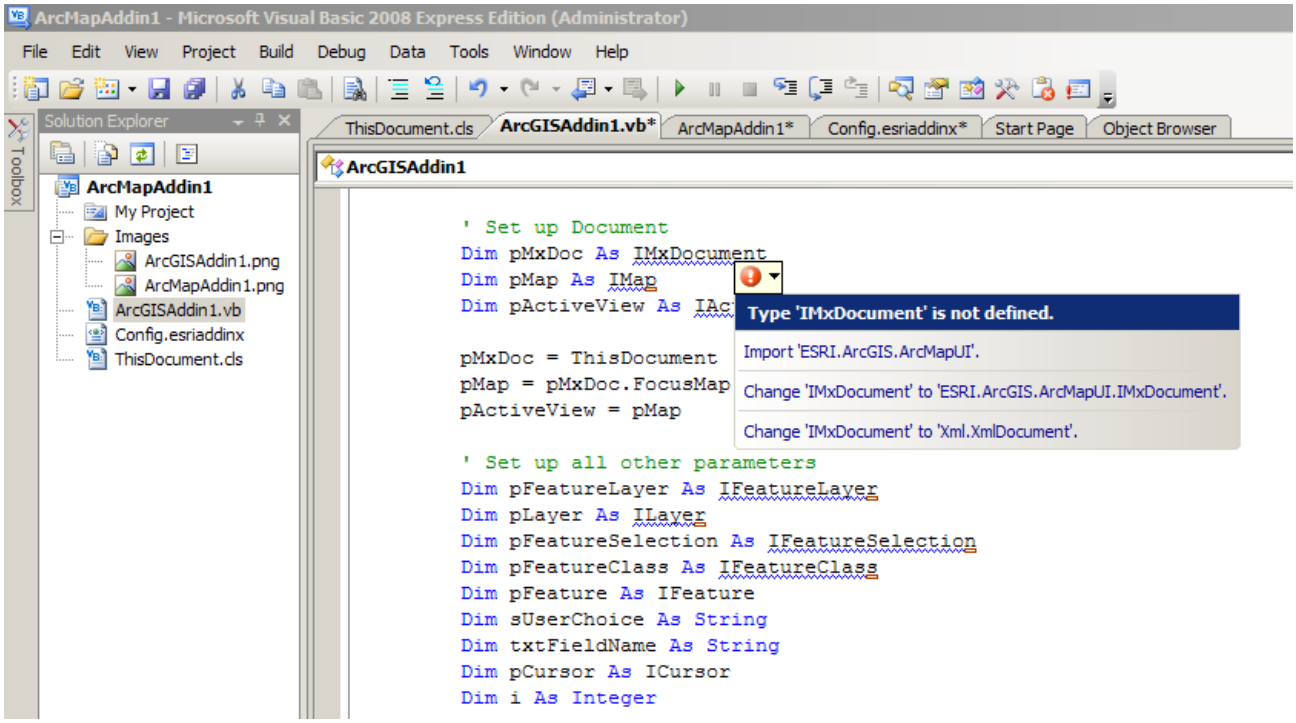

Figure 2. Context-sensitive error suggestions can be helpful to quickly correct errors. In this case, MSVS recognizes that in order for the "IMxDocument" interface to work properly, a reference to 'ESRI.ArcGIS.ArcMAPUI' must be added to the code. Unfortunately, not all reference errors are recognized automatically and will require the user to search the Help or Object Browser for a solution.

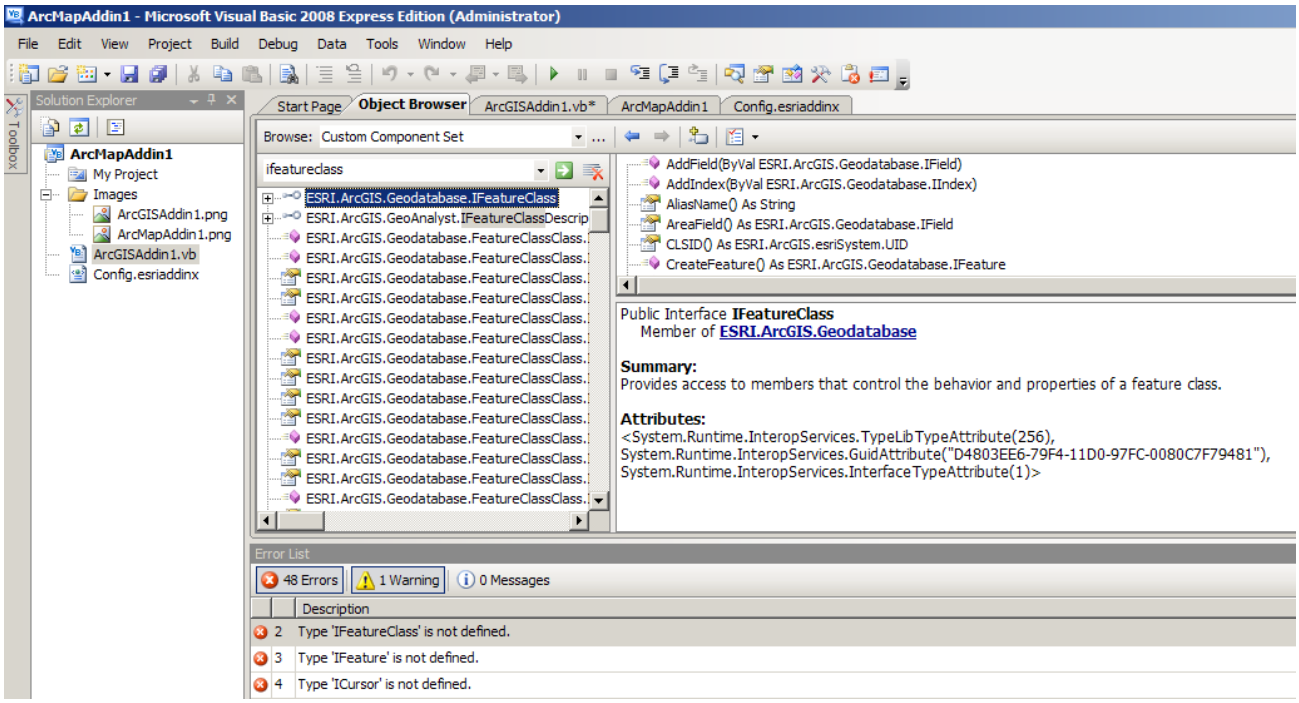

Figure 3. The Error List shows that "IFeatureClass" is not defined. Search the Object Browser for interfaces that are not recognized by the context-sensitive error handling. A search for "IFeatureClass" returns information about its functions, and a reference to its parent member "ESRI.ArcGIS.Geodatabase", to which a reference must be added for the interface to be accessible by the current project.

Fixing Some Common Errors

Errors in code that has been imported from VBA to VB.NET are inevitable, with varying levels of complexity when it comes to identifying and correcting their causes. In the process of unifying the SDK for ArcGIS, Esri has necessarily updated and changed some components within ArcObjects, which will in turn create errors in your code where before there were none. While this paper is not intended to be a comprehensive troubleshooting guide, it does discuss some very common errors that have simple solutions, which may be helpful to anyone making the transition from VBA to VB.NET.

Before trying to correct your code, remember that Add-ins and VB.NET are fundamentally different from VBA customizations in many ways but that the most important thing to do when converting code is to determine which object class references your project will need. In VBA, objects could be referenced implicitly; that is, every object that VBA could possibly access was exposed and could be called without adding references to specific classes of ArcObjects. In VB.NET (and others), object class references are always explicit and you will need to expose the object classes you plan to use in your project. These references must be assigned to your project in two ways: (1) at the project level and (2) in the code. It is important to understand the need for both sets of references. At the project level, the references are needed for the MSVS code editor to give context-sensitive help and debugging and to properly register the Add-in within the application framework when it is built (compiled). In the code, "Imports" statements provide hooks for the editing environment so that functions within each imported class are available implicitly within the current project, eliminating the need for additional syntax to make them explicit:

With an "Imports" statement placed *before* the first "Class" statement:

```
Imports ESRI.ArcGIS.Carto
```

… dimensioning can be done *implicitly*:

```
Dim pFeatureLayer As IFeatureLayer
```

… *instead* of explicitly:

```
Dim pFeatureLayer As ESRI.ArcGIS.Carto.IFeatureLayer
```

One of the most common errors that will appear in the converted code for a button or tool is the reference that most VBA projects use to hook into the open, currently active ArcMap document. Typically dimensioned as "pMxDoc" or similar, this reference was set to be equal to "ThisDocument." Since VBA was integrated into the application framework, this and other references to the active or open application window or document were coded without an explicit object reference. The fix for this is very easy. Simply change "ThisDocument" to "My.ArcMap.Document" and your project will have the correct, explicit reference to the parent application and its currently active document:

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
```

…becomes:

```
Dim pMxDoc As IMxDocument = My.ArcMap.Document
```

Also note the slight change in syntax between the old and new references: the "Set" statement has been dropped and the lines are combined into one statement that simultaneously defines and assigns a value to the object "pMxDoc." This is due to a minor but important change in the syntax of assigning values to variables, which leads us to the next most common error: "Set" statements.

In VBA and VB6, many objects had a default property. This required that the "Set" statement be used in order to differentiate between the definition of the object *reference* and the *default property* of the object. With the removal of default properties of objects in VB.NET, the use of "Set" statements has become obsolete, and so too the need for a separate line to "set" a reference to an instance of an object. In most cases, simply going through your code and deleting all instances of the "Set" keyword should correct most errors, but some will persist in situations where the default property was used. This will require the removal of the "Set" statement as well as the addition of an explicit property keyword:

```
Dim lbl1 As Label, lbl2 As Label
lbl1 = "Label 1"     ' Assign value to lbl1's default property (Caption)
lbl2 = lbl1          ' Replaces lbl2's default property with lbl1's
Set lbl2 = lbl1      ' Replace lbl2 with an object reference to lbl1
```

…becomes:

```
Dim lbl1, lbl2 As New Label       ' Both become type Label
lbl1.Text = "Label 1"       ' EXPLICITY define the Text property
lbl2.Text = lbl1.Text       ' Copy Text property from lbl1 to lbl2
lbl2 = lbl1                 ' Copy object reference, "Set" not required
```

In a way, the object reference itself is now the default property, with all other properties becoming explicit.

The previous example also highlights another common source of errors; those brought about by the elimination of default properties. When converting from VBA, since default properties were allowed, your code probably contains at least a few of these errors. You might also find that some properties have changed name or been eliminated. See the following examples:

In VBA, this code worked:

```
'--- Define the unique identifier for geofeature layers
pGFL_UID = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"
```

… but in VB.NET, a warning is issued about type conversion:

"Runtime errors might occur when converting 'String' to 'ESRI.ArcGIS.esriSystem.UID'."

… because the object "pGFL_UID" needs its "Value" property set explicitly:

```
pGFL_UID.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"
```

In VBA, the "Caption" property was a default property for several objects related to the construction of user forms. This property has been eliminated and replaced by the "Text" property. Labels on forms that were defined, even explicitly, will need to be corrected:

```
lbl1 = "Label 1"                ' Throws an error...
lbl1.Caption = "Label 1"   ' So does this...
lbl1.Text = "Label 1"           ' This one works!
```

In the same vein, you may get a warning that there is an object "passed by reference before it has been assigned a value. A null reference exception could result at runtime." To avoid this error, you can set an object reference equal to "Nothing" until it is time to define it properly in your code:

```
Dim m_pEnumGxObject As IEnumGxObject          ' Gives a warning
```

… whereas:

```
Dim m_pEnumGxObject As IEnumGxObject = Nothing ' No warning!
```

This type of error rarely results in the code not functioning correctly. It is merely good practice to get in the habit of assigning object references explicitly in your code.

New Methods of Error Handling

With the potential for the number of errors being high in the initial conversion, it is very helpful not only to use the MSVS Error List to help find errors and correct them but also to update the error handling in your code. In VBA, it was common to use the "On Error GoTo …" statement to catch errors in code. This code construct is no longer supported, so you will need to update your error handling. The "Try…Catch…Finally" construct is now the preferred method for error handling. It has the advantage of being able to deal with unhandled (unexpected) errors while also allowing you to decide what errors to handle explicitly with customized error messages and actions:

```
Public Sub Example()

 Try
 ' Code to try and set a value for pObject goes here, then check it

      If pObject Is Nothing Then
        ' Handle this object definition error explicitly
        ' Pass this message to the "Catch" statement
      Throw New Exception("Error defining pObject.")
      Else : End If

 ' If all is well with pObject, code continues here...

 Catch ex As Exception     ' This catches all errors
      ' If the exception was handled, displays message you created
      MsgBox("An exception has occurred: " & ex.Message)
 Finally
      ' Put more code here to execute after error is handled
 End Try

 End Sub
```

This is a very simple example of an extremely powerful code construct. For additional information regarding the "Try…Catch…Finally" statement, see the *MSVS help*.

Implementing User Forms Within an Add-in

Many customizations to ArcGIS require a user form to get input from, or display information to, the user. User forms in .NET have some very different behavior than they did in VBA. One major problem when upgrading VBA user forms to .NET is that, unlike code modules, a form's design/layout module is not importable and will need to be reconstructed in MSVS. But, since much of the functionality and code has to be "rewired" anyway, redesigning the form is just a necessary inconvenience. Depending on the complexity of the form that is being upgraded, or if your customization uses several linked forms, recreating the functionality and behavior can be tricky in .NET. In these cases, see the Help and Resources section below; there are many Web resources for help in making the switch if you are facing a more complicated scenario. However, there are several basic behavioral and code-related issues with forms in .NET that are important to understand in order to make it easier to use a form in your Add-in.

The most significant difference between user forms in VBA and .NET is that in VBA the form object was implicitly referenced throughout the project; that is, once the form was created in the VBA project, it could be called upon without further dimensioning or instantiation. As with other objects in .NET, the form object must be instantiated and defined *explicitly* in order to function correctly. Because of this change, the way in which forms are handled and referenced across the project is also quite a bit different, especially if there are multiple forms that must interact with one another. MSVS help has an excellent article on using forms in VB.NET, how to use multiple forms, and how to upgrade form-calling syntax used in VBA and VB6 (see *http://msdn.microsoft.com/en-us/library/aa289529(VS.71).aspx*).

User forms in VBA were "modal" by default. Modal forms are displayed to the user with the execution of code following the call effectively paused until the form is hidden or closed. A form within an Add-in created in .NET has much more flexibility in its use, display options, and behavior. If your application's form must be modal as it would have been in VBA, you will need to be specific when displaying the form using the "FormName.ShowDialog" construct. Another useful option in the form properties is the "FormName.TopMost" switch. This option allows the form to ride above all other forms, while giving the user the ability to access other parts of the application without closing the form. A common example of a form of this type is a Find and Replace tool window. More information about displaying forms in Add-ins and VB.NET can be found in the MSVS help topic *Form Class*.

Conclusion

It is my hope that this article will be helpful to those beginning the transition from VBA to the new Add-in model for customizing ArcGIS. By adopting the Add-in model, Esri has greatly expanded the ability of users to customize their products with tools and user interfaces that were not previously available. While this article focuses on the VB.NET approach, there are many resources for developing in all the major languages available from Esri and Microsoft. Below, I provide some links to the most important online resources, but do not forget that a little creative Web searching can also provide answers to your questions. Chances are, someone has had the same problem or asked the same question you are currently pondering!

Help and Resources for Customizing ArcGIS 10

These resources will aid in converting VB6/VBA code to VB.NET (or other languages), as well as help you create customizations to ArcGIS 10 using add-ins. I have also included links to legacy topics. Much of the information stored on the legacy sites can still be quite useful:

ArcGIS 10 Web Help:
- Searchable help for all aspects of the ArcGIS 10 software package.
http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html

Esri Resource Center:
- ArcGIS 9.3 (and later) resources for developers. Includes information on configuring the user interface, using Python for geoprocessing, writing add-ins for ArcGIS Desktop, and the use of the comprehensive ArcObjects library for developing custom software and extensions.
http://resources.arcgis.com/content/arcgisdesktop/10.0/customizing

- ArcObjects .NET API Code Gallery – successor to ArcScripts.
http://resources.arcgis.com/gallery/file/arcobjects-net-api

Esri Support Center:
- Search for help with solutions to automation problems. User can create a free Esri Global Account and post questions, watch threads, and post solutions to others' problems. The Global Account also provides access to free webinars and other exclusive training materials relating to ArcGIS. Highly recommended!
http://support.esri.com

MS Visual Studio Help (2008):
- Links to topics relating to all things MSVS.
http://msdn.microsoft.com/en-us/library/52f3sw5c(VS.90).aspx

Legacy Help Sites (ArcGIS 9.3 and earlier)

Esri Developer Network:
- Home page for licensing developer tools, resource center, and developer community pages. Links to version 9.2 and prior development resources still available here.
http://edn.esri.com

- Code Exchange – find code samples and documentation for ArcGIS 9.2 and earlier.
http://edn.esri.com/index.cfm?fa=codeExch.gateway

Getting started with VBA:
- "Getting started with VBA" in the ArcGIS 9.3 Desktop Help.
http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Getting_started_with_VBA

- "Sample VBA Code" in the ArcGIS 9.3 Desktop Help.
http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Sample_VBA_code

- "Customizing ArcGIS [9.3] Desktop with VBA".
http://resources.esri.com/help/9.3/arcgisdesktop/com/vba_start.htm

Esri ArcScripts:
- Home page for user community script posting and exchange. This site has been closed to new postings since April 2010, but all content is still searchable and downloadable. Many useful scripts for version 9.3.1 and earlier. Search with keyword "Wunderlich" to find my scripts from the DMT'09 presentation.
http://arcscripts.esri.com

Acknowledgments

Support by Dr. Robert D. Hatcher, Jr. (University of Tennessee Science Alliance Centers of Excellence) and the USGS National Cooperative Geologic Mapping Program, through Grant Number G10AC00001, is greatly appreciated.

References

ArcGIS Resource Center, 2011a, A quick tour of Python: Desktop Help 10.0, accessed at *http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//002z00000023000000.htm*.

ArcGIS Resource Center, 2011b, Building add-ins for ArcGIS Desktop: ArcObjects SDK 10 Microsoft .NET Framework, accessed at *http://help.arcgis.com/en/sdk/10.0/arcobjects_net/conceptualhelp/index.html#/Building_add_ins_for_ArcGIS_Desktop/0001000000w2000000/*.

ArcGIS Resource Center, 2011c, ArcObjects SDK 10 system requirements: ArcGIS SDKs 10, accessed at *http://resources.arcgis.com/content/arcgissdks/10.0/system-requirements*.

Wunderlich, A.L., 2010, Improving ArcGIS workflow: Automation using Visual Basic for Applications (VBA), *in* Soller, D.R., ed., Digital Mapping Techniques '09—Workshop Proceedings: U.S. Geological Survey Open-File Report 2010–1335, p. 21-26, *http://pubs.usgs.gov/of/2010/1335*.