

Customization in ARCGIS

Two levels of Customization

- Customizing the Interface
 - To create more efficient user interfaces
- Grouping frequently used tools
- Create specialized applications
 - To access capabilities not on the standard interface
- Supplied as part of ArcGIS but not on the standard interface (many!)
 - Downloaded from the web
 - » From ESRI web site, supplied by ESRI or users
 - » From other web sites
 - Developed by you
- Developing Additional Capabilities
 - Automating repetitive tasks
 - Creating new analytical procedures

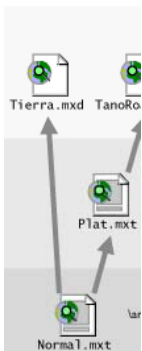
Customizing ArcMap

You can customize ArcMap by:

- adding references to geographic data and define how the data is displayed.
- creating map layouts with spatial reference and ancillary cartographic elements.
- adding, removing, or rearranging elements of the standard user interface.
- writing code in a Visual Basic for Applications project.

Customizing ArcMap

All customization in ArcMap is stored in a map document or a map template.



Map documents are based on templates. They can be based on a project template or directly on the normal template

Project templates can be created anywhere on your file system. Changes made to the project templates propagate to all documents based on that template.

There is always a normal template (stored in arcexe/bin/Templates folder). Any changes to the normal template propagate to all templates and documents

Documents and Templates

Whenever you are using ArcMap you are working with a document referred to as a map.

This document stores the map state, the state of the user interface, custom user interface settings, a Visual Basic for Applications project, and other application-specific information, such as cartographic layouts.

A template is a kind of map document that is specified to be a starting point for a new map document. A template allows for sharing customization.

Documents and Templates

When you make customization changes to the user interface in ArcMap, you save these changes into a document of template. You can save to:

The current map document: There is always a map document open in ArcMap.

A base template: Templates may contain any of the following: data, a custom interface, and a predefined layout that arranges map elements, such as north arrows, scale bars, and logos, on the virtual page. ArcMap templates have a .mxt file extension. There may not always be a base template loaded in ArcMap.

The Normal template: A special template that is automatically loaded in ArcMap. This template stores any personal settings you have made to the user interface that you want loaded every time you use ArcMap.

Since any modifications made to Normal will be reflected every time you create or open a document, you should be careful when making changes to Normal.

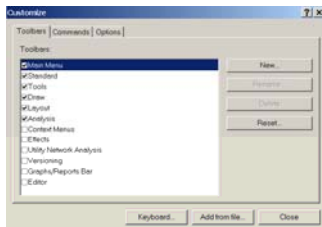
Customizing the Interface

You can use existing toolbars or you can create new toolbars to organize commands that you often use together or to contain buttons that run custom scripts.

The Customization dialog box is used to create new toolbars and add or remove controls.

Customization Dialog Box

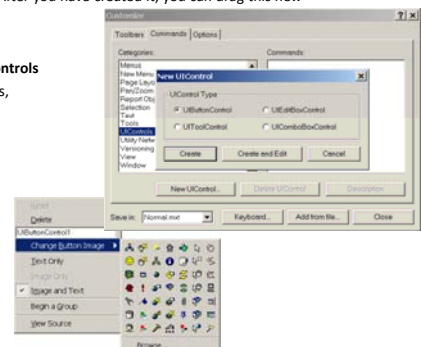
- A graphic interface for customization. To open
 - Go to *Tools>Customization*
 - Or, double click on empty area in a toolbar
- Modify user interface
 - Turn toolbars on and off
 - Create new toolbars
 - Create new controls
 - Add, delete, and move controls (buttons and menus)
 - Manage shortcut keys



UI Controls

You can create a new button, tool, combo box, or edit box (collectively called UIControls), then attach code to the control's events, such as what happens when you click a button. After you have created it, you can drag this new control onto a toolbar

- **User Interface (UI) Controls**
 - UIButtons, UItools, UIEditBoxes, and UIComboBoxes
- **Control Properties**
 - Text or no text
 - Image
 - Caption
 - Group



The User Interface Components

Menus arrange commands into a list. A context menu is a floating menu that pops up at the location of the pointer when you right-click.

Buttons and menu commands run programming code when you click them.

Tools require interaction with the display before an action is performed—that is, before their programming code is run. The Zoom In tool is a good example—you click or drag a rectangle over a map before seeing its contents in more detail.

Combo boxes allow you to choose an option from a drop-down list.

Text boxes or edit boxes allow you to type in text. In ArcMap, you can type the scale at which you want to view the map.



Creating custom commands with VBA.

Visual Basic Applications is not a standalone program; it's embedded in the applications. It provides an integrated programming environment, the Visual Basic Editor (VBE), which lets you write a Visual Basic (VB) macro, then debug and test it right away in ArcMap.

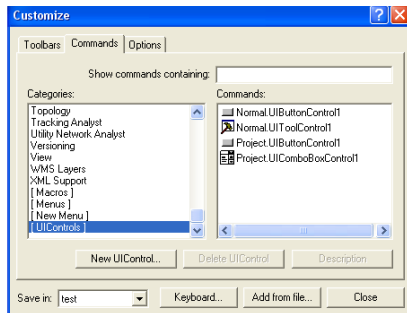
In ArcMap each currently loaded document and template has an associated VBA project in the VBE, allowing you to write macros that are specific to a given map document or to apply macros to all the documents consuming a specific template.

The VBA project for the document is called Project followed by the name of the document in brackets. For example, in ArcMap it is named Project (<name of document>.mxd), and the VBA project for the Normal template is called Normal (Normal.mxt).

Creating custom commands with VBA.

When you create a custom component you can save it in the Normal.mxt or within an ArcMap document (.mxd). If you save it within the .mxd, it is only available when you use that .mxd file.

The Save in combo box lists the names of the currently loaded document and templates. In ArcMap, use this setting to specify whether the change you are about to make will be saved in the document, the Normal template, or another template. The default setting is the Normal template.



If you change the interface and want to return to the original settings:

Simply delete the Normal template file. Upon startup, ArcMap and ArcCatalog will regenerate the Normal templates if they are missing.

The ArcCatalog Normal template, Normal.gxt, is located in user settings. For example, in Windows 2000 or XP, it can be found in C:\Documents and Settings\User\Application Data\ESRI\ArcCatalog.

The ArcMap Normal template, Normal.mxt, is also located in user settings. For example, in Windows 2000 or XP, it can be found in C:\Documents and Settings\User\Application Data\ESRI\ArcMap\Templates.

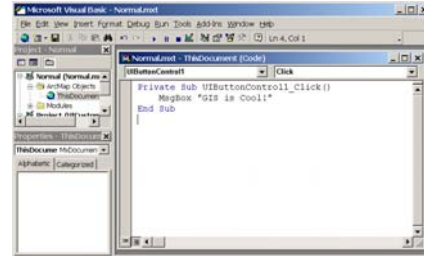
Visual Basic Editor

You can open the VBE in any of the above-listed applications by clicking Tools on the Main menu, pointing to Macros, then clicking Visual Basic Editor.



Visual Basic Editor

- Tools> Macros> Visual Basic Editor
- View Source



Getting started with VBA

To begin programming with VBA in ArcMap, start the Visual Basic Editor.

To start the Visual Basic Editor

Start ArcMap

Click the Tools menu, point to Macros, then click Visual Basic Editor. You can also use the shortcut keys Alt+F11 to display the Visual Basic Editor. To navigate among the projects in the Visual Basic Editor, use the Project Explorer. It displays a list of the document's modules, class modules, and user forms.

To add a macro to a module

ArcMap provides a shortcut for creating a simple macro in a module.

Click the Tools menu, point to Macros, then click Macros.

Type the name of the macro you want to create in the Macro name text box. If you don't specify a module name, the application creates a module called *modul_{xx}* and stores the macro in that module. If no module is specified after you specify a module, and a module is already active, the macro is placed in that module. Preceding a macro's name with a name and a dot stores it in a module with the specified name. If the module doesn't exist, the application creates it.

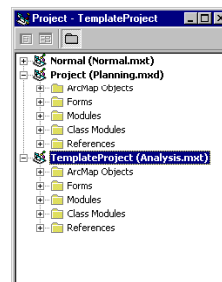
Click the dropdown arrow of the Macros in the combo box and choose the VBA project in which you want to create the macro.

Press the Enter key or click Create.

The stub for a Sub procedure for the macro appears in the Code window.

Visual Basic Editor

ArcMap has a default project associated with its document that's listed in the Project Explorer as Project followed by its filename. In addition, you'll see another project listed in the Project Explorer called Normal (Normal.mxt).



Saving your customizations in a template

Create a new map document.

Make all the changes to the user interface you want.
When you make any of your changes to the user interface, make sure that the document name is selected in the drop-down Save in combo box

Click the File menu and click Save As.

Click the Save as type drop-down arrow and click ArcMap Templates (*.mxt).

Navigate to the folder where you want the template saved. The template can be saved in any folder on your network.

Type a new name for the new template. You can't use Normal.mxt as a template name; that name is reserved for the Normal template.

Click Save

Creating macros

When you create a macro, you're creating a VB Sub procedure.

The procedure's name is the name you assign to the macro.

You add code to the procedure in a Code window just as you would in VB.

When you create a new macro on the Macros dialog box, precede the macro's name with the name of the module in which to store it. You can organize your macros in different modules; each module has its own Code window.

To add your macro to a specific module, type the module name before the macro's name, for example, "Department.WorkMacro". If the module doesn't exist, a new module with that name is created for you and added to the VBA project. Similarly, if you provide a name for a new macro but don't specify which module to store it in, a new module, NewMacros, is created.

Creating macros

Click the Tools menu, point to Macros, then click Macros.

Click the drop-down arrow of the Macros in combo box, then click the document or template in which you want to create the macro.

Type the name of the macro you want to create in the Macro name text box. Preceding the name of the macro with a module's name and a dot stores the macro in the specified module. If you don't specify a module name, the application stores the macro in a module named NewMacros.

Press Enter or click Create.
The stub for a Sub procedure for the macro appears in the Code window.

Type the code for the macro between the first line (Sub *name*()) and last line (End Sub).

Click the VBE File menu and click Save Project.

Running a macro from a Visual Basic module

In the Visual Basic Editor, open the Code window where the macro's code is saved.

In the Code window, click inside the Sub procedure of the macro you want to run.

Click the Run menu and click Run Sub/UserForm.

Running a macro in the Macros dialog box
Click the Tools menu, point to Macros, then click Macros.

Click the Macros in drop-down arrow and click the document or template containing the macro you want to run.

Type the name of the macro you want to run or click its name in the list.

Click Run.

Creat custom commands with VBA

Click the View menu, point to Toolbars, then click the toolbar to which you want to add a new command.

Click the Tools menu and click Customize.

Click the Commands tab.

Click the drop-down arrow on the Save in combo box, then click the document or template in which the new command will be saved.

Click [UIControls] in the Categories list.

Click New UIControl.

Click the type of UIControl you want to create.

Click Create to create the control without attaching code to it. The name of the control appears in the Commands list. You can add code for the control at another time. If you want to start adding code to the control right away, click Create and Edit.

Click the newly created UIControl, click it again to activate in-place editing, then type a new name for the control.

Click and drag the newly created UIControl and drop it on a toolbar or menu.

On the toolbar or menu, right-click the command to set its image, caption, and other properties.

Right-click the new command on the toolbar or menu, then click View Source. The Visual Basic Editor appears, displaying the control's code in the Code window.

Click the Procedures/Events drop-down arrow and click one of the control's event procedures.

Type code for the event procedure.

Repeat until all the appropriate event procedures have been coded.

Click Save on the Visual Basic Editor.

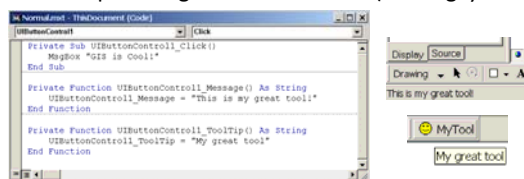
Click the Close button on the Visual Basic Editor.

If you clicked Create and Edit in step 8, open the Customize dialog box, click the Commands tab, and drag the newly created UIControl from the Commands list to a toolbar or menu.

Click Close on the Customize dialog box.

ToolTips

- Help for your controls
- Set the Tooltip property
- Help message in the status bar (Message)



Object library

ArcGIS is component object model (COM)-based, and the system is built and extended using the ArcObject software components.

ArcObjects are an integrated collection of cross-platform GIS software components that are client/server ready. The ArcGIS family of applications relies on ArcObjects to provide data management, map presentation functionality, and more.

There is an extensive ArcObjects object library available to help you customize applications in ArcGIS Desktop. The [ESRI Developer Network \(EDN\)](#) describes the classes, interfaces, properties, methods, and enumerations that are available

If you are using VBA inside ArcMap or ArcCatalog, most of the common ESRI object libraries are already referenced for you.

Your code manipulates the objects by getting and setting properties on their interfaces, such as setting the *MaximumScale* and *MinimumScale* of a map's *FeatureLayer*; invoking methods on the interfaces, such as adding a vertex to a polyline; or setting a field's value. The code runs when an event occurs, for example, when a user opens a document, clicks a button, or alters data by modifying an edit sketch.

In the VBA development environment you can add modules, class modules, and user forms to the default project contained in every ArcGIS application document.

A project can consist of as many modules, class modules, and user forms as your work requires.

A project is a collection of items to which you add code. A module is a set of declarations followed by procedures—a list of instructions that your code performs. A class module is a special type of module that contains the definition of a class, including its property and method definitions. A user form is a container for user interface controls, such as command buttons and text boxes.

Adding modules and class modules

All ArcGIS application documents contain the class module *ThisDocument*, a custom object that represents the specific document associated with a VBA project. The document object is called *MxDocument* in ArcMap and *GxDocument* in ArcCatalog. The *IDocument* interface provides access to the document's title, type, accelerator table, command bars collection, parent application, and Visual Basic project.

Adding modules and class modules

Once you've invoked the Visual Basic Editor, you can insert a module, class module, or user form.

Then you insert a procedure or enter code for an existing event procedure in the item's Code window, where you can write, display, and edit code.

You can open as many Code windows as you have modules, class modules, and user forms, so you can easily view the code and copy and paste between Code windows. In addition to creating your own modules, you can import other modules, class modules, or user forms from disk.

Adding modules and class modules

Modules and class modules can contain more than one type of procedure: sub, function, or property. You can choose the procedure type and its scope when you insert a procedure. Inserting a procedure is like creating a code template into which you enter code.

Every procedure has either private or public scope. Procedures with private scope are limited to the module that contains them—only a procedure within the same module can call a private procedure. If you declare the procedure public, other programs and modules can call it.

Variables in your procedures may either be local or global. Global variables exist during the entire time the code executes, whereas local variables exist only while the procedure in which they are declared is running. The next time you execute a procedure, all local variables are reinitialized. However, you can preserve the value of all local variables in a procedure for the code's lifetime by declaring them static, thereby fixing their value.

To add a procedure to an existing module

In the Project Explorer, double-click the ArcMap Objects or Modules folder, then choose the name of a module. Ensure that the code view of the module is active by clicking the View Code button.

Click the Insert menu and click Procedure.



Type the name of the procedure in the Name text box.

Click the Type dropdown arrow and click the type of procedure: Sub, Function, or Property. Click the Scope dropdown arrow and click Public or Private.

To declare all local variables static, check the All Local variables as Statics check box. Click OK. VBA stubs in a procedure into the item's Code window into which you can enter code. The stub contains the first and last lines of code for the type of procedure you've added. Enter code into the procedure.

For interface references, declare an interface variable and use the *Set statement* to assign the interface reference to the property.

For other values, declare a variable with an explicit data type or use Visual Basic's *Variant data type*. Then, use a simple assignment statement to assign the value to the variable.

Properties that are interfaces can be set either by reference or by value.

Properties that are set by value do not require the *Set statement*.

```
Dim pEnv As IEnvelope
Set pEnv = pActiveView.Extent 'Get extent property of view.
pEnv.Expand 0.5, 0.5, True 'Shrink envelope.
pActiveView.Extent = pEnv 'Set By Value extent back on
IActiveView.
Dim pFeatureLayer as IfeatureLayer
Set pFeatureLayer = New FeatureLayer 'Create New Layer.
Set pFeatureLayer.FeatureClass = pClass 'Set ByRef a class into
layer.
```

Using the Global Application objects

Application and *ThisDocument* are examples of global system variables that can be accessed by any module or class in the VBA environment while ArcMap is running. This variable is automatically set to reference the current document when ArcMap opens the document. You can use *ThisDocument* as a shortcut when programming in VBA to access the current document. Here is an example of how to use both the *Application* and *ThisDocument*:

```
Dim pMxDoc as IMxDocument
Set pMxDoc =Application.Document 'or
Set pMxDoc =ThisDocument
```

You can use *ThisDocument* as a shortcut when programming in VBA to access the current document.

Here is an example of how to use both the *Application* and *ThisDocument*:

```
Dim pMxDoc as IMxDocument
Set pMxDoc = Application.Document
'or
Set pMxDoc = ThisDocument
```

Both methods illustrated above result in a reference being set to the local document.

Running VBA code

As you build and refine your code, you can run it within VBA to test and debug it. For more information about running and debugging a VBA program, such as adding breakpoints, adding watch expressions, and stepping into and out of execution, see Microsoft Visual Basic online help.

To run your code in the Visual Basic Editor or from the Macros dialog box

Click the Tools menu and click Macros.

In the Macro list, click the macro you want and click Run.

If the macro you want is not listed, make sure you've chosen the appropriate item: either Normal, Project, or TemplateProject in the Macros In box. Private procedures do not appear in any menus or dialog boxes.

To run only one procedure in the Visual Basic Editor

In the Project Explorer, open the module that contains the procedure that you want to run.

In the Code window, click an insertion point in the procedure code. Click the Run menu and click Run Sub/UserForm.

Saving a VBA project

To save your ArcMap document and your VBA project, click Save from the ArcMap File menu or Save <File Name> from the File menu in the Visual Basic Editor. Both commands save your file with the project and any items stored in it. After saving the file, its filename is displayed in the Project Explorer in parentheses after the project name. To save the document as a template, click Save As from the ArcMap File menu and specify ArcMap Templates (*.mxt) as the File type.

Adding user forms

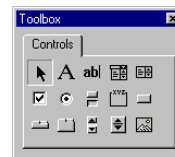
If you want your code to prompt the user for information, or you want to display the result of some action performed when the user invokes an ArcGIS application command or tool, or in response to some other event, use VBA's user forms.

User forms provide a context in which you can provide access to a rich set of integrated controls. Some of these controls are similar to the UIControls that are available as part of the Customize dialog box's Commands tab. In addition to text boxes or command buttons, you have access to a rich set of additional controls. A user form is a container for user-interface controls, such as command buttons and text boxes. A control is a Visual Basic object you place on a user form that has its own properties, methods, and events. You use controls to receive user input, display output, and trigger event procedures. You can set the form to be either modal, in which case the user must respond before using any other part of the application, or modeless, in which case subsequent code is executed as it's encountered.

To add and start coding in a user form

In the Project Explorer, select the Project to which you want to add a user form. Click the Insert menu and click UserForm.

VBA inserts a user form into your project and opens the Controls Toolbox.



Click the controls in the Controls Toolbox that you want to add to the form.

Add code to the user form or to its controls.

To display the Code window for a user form or control, double-click the user form or control. Then, choose the event you want your code to trigger from the dropdown list of events and procedures in the Code window and start typing your code. Or, just as in a module or class module, insert a procedure and start typing your code.

To display the form during an ArcMap session in response to some action, invoke its Show method, as in this example:

```
UserForm1.Show vbModeless 'show modeless
```

You can reset core commands:
 Click View, point to Toolbars, and check to show the toolbar with the command you want to reset.

Click the Tools menu and click Customize.

On the toolbar, right-click the command you want to change.

Click Reset.
 The command returns to its default settings.

Resources

[Getting to Know ArcObjects: Programming ArcGIS with VBA](#)

As much an introduction to VB as it is an introduction to ArcObjects
 Focus on GUI, vector programming tasks. Doesn't get into much detail
 on complex analytical solutions (i.e. spatial analysis)

[Exploring ArcObjects](#) - Vol I and II

More reference manuals than tutorials

Must have decent VB knowledge and preferably a little ArcObjects
 practice before picking this up.

[ArcScripts](#)

```
Private Function UIButtonControl1_Checked() As Boolean
UIButtonControl1_Checked = False
End Function
```

```
Private Sub UIButtonControl1_Click()
Dim pMxDoc As IMxDocument
Dim SelCount As Long
Set pMxDoc = Application.Document
SelCount = pMxDoc.FocusMap.SelectionCount
MsgBox SelCount
End Sub
```

```
Private Function UIButtonControl1_Enabled() As Boolean
Dim pMxDoc As IMxDocument
Dim LayerCount As Long
Set pMxDoc = Application.Document
LayerCount = pMxDoc.FocusMap.LayerCount
If LayerCount > 0 Then
UIButtonControl1_Enabled = True
Else
UIButtonControl1_Enabled = False
End If
End Function
```

```
Private Function UIButtonControl1_Message() As String
UIButtonControl1_Message = "Return selection count for all
layers"
End Function
```

```
Private Function UIButtonControl1_ToolTip() As String
UIButtonControl1_ToolTip = "Selection Count"
End Function
```