CHAPTER 5: VBA OVERVIEW

Training Objectives

Actively participating during this chapter helps you learn:

- The Microsoft Dynamics™ GP VBA Components.
- About Methods, Properties, and Events.
- How to Build a User Form.
- How to Declare Variables.
- How to Compile and Debug within VBA.

Introduction

Microsoft® Visual Basic for Applications® (VBA), is a standard programming language used by Microsoft Office products, and a number of software providers who embed or host VBA. Microsoft Dynamics GP hosts VBA when you install and register Modifier Release 4.0 or later. The tools available in the VBA environment allow you to customize Microsoft Dynamics GP windows and reports. In addition, you can attach VBA code to new fields created using Modifier.

Supported VBA 6.0 Features

Add-Ins

VBA offers add-ins, allowing modifications to use custom-built user interfaces and tools, and it provides an efficient method for reusing customization code.

Modeless Windows

New VBA user forms have an additional form property: ShowModal. When set to False, it is possible to return to a Microsoft Dynamics GP window without closing the VBA form, extending the usability of new user forms. Note that although focus can leave the User Form, it does remain the active, or front most form. This limitation can be addressed by writing code to manipulate the opening position of the windows.

Full Core-Language Parity with VB 6

Enhancements in VB 6 are shared by VBA, including:

- Improved code security
- New string manipulation functions
- Custom dialog boxes

Additional References

VBA is a powerful programming language, and a basic knowledge of VBA programming concepts is required. If you have not worked with VBA, study or review one or more of the additional learning tools available for purchase before using VBA with Microsoft Dynamics GP:

Microsoft Office 2000/Visual Basic for Applications Fundamentals
By David Boctor, Microsoft Press
Published, May 1999

Microsoft Visual Basic 6.0 Programmer's Guide
By Microsoft Press
Published, June 1998

VBA Developer's Handbook
by Ken Getz and Mike Gilbert
Published 2001

VBA for Dummies
By Steve Cummings
Published, May 2001

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

Microsoft Dynamics GP VBA Components

When using VBA with Microsoft Dynamics GP, the following components are important to understand:

- The Visual Basic Editor
- Microsoft Dynamics GP Objects
- The Microsoft Dynamics GP Project
- Properties, Methods, and Events
- User Forms
- Debugging
- Variables

The Visual Basic Editor

The Visual Basic Editor is included in Microsoft Dynamics GP allowing you to build and maintain a Microsoft Dynamics GP VBA project. It becomes available when Microsoft Dynamics GP Modifier is registered. Select the Visual Basic Editor by TOOLS→CUSTOMIZE→VISUAL BASIC EDITOR.

The following illustration shows how the Visual Basic Editor looks when you access it from Microsoft Dynamics GP for the first time. This will vary depending on the dictionaries that exist for the Microsoft Dynamics GP installation.

Because the Visual Basic Editor is identical between host applications, if you have used VBA with Microsoft® Excel or Microsoft® Word, the Visual Basic Editor for Microsoft Dynamics GP is the same. The following is a list of the components of the Visual Basic Editor along with a brief description of each.

Component	Description
Project Explorer	Shows each of the elements of your VBA project. For example, within the Microsoft Dynamics GP project, it includes all of the Microsoft Dynamics GP windows and reports added to your project as well as any new VBA forms and modules you may have created.
Code Window	Use to write any VBA code associated with your project.
Form	Displays the layout of your custom VBA forms.
Toolbox	Provides a set of controls used on your custom forms.
Properties	Use to review and set properties for the objects in your project such as changing the background color of a field on one of your forms.

Microsoft Dynamics GP Objects

Microsoft Dynamics GP exposes its functionality to VBA through a set of programmable objects. These objects include:

- Windows
- Reports
- Grids
- Dynamics User Object Store (DUOS) objects

The relationship each object has to another object is expressed in a hierarchical object model, shown in the following illustration:

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

The organization of this model is important. You must navigate through the object model within VBA code to access lower-level objects. For example, to access a window field, first indicate the field's window object:

```
'Set a field on a window
CustomerMaintenance.CustomerID.Value = "AARONFIT0001"
```

More detail is given on each of the Microsoft Dynamics GP objects later in this manual.

The Microsoft Dynamics GP Project

Using the Visual Basic Editor, all of the customizations done for Microsoft Dynamics GP objects are stored in a project file called DYNAMICS.VBA. This file is created the first time you open the Visual Basic Editor from within Microsoft Dynamics GP and is created in the same location as the Microsoft Dynamics GP application.

NOTE: *If working with a third-party application, VBA creates a separate project to store customizations for those objects. The project file is named PRODUCT.VBA, where PRODUCT is the name of the third-party product's application dictionary. This file is also created in the same location as the Microsoft Dynamics GP application and is created the first time you open the Visual Basic Editor for the third-party product.*

Objects defined for the Microsoft Dynamics GP VBA project appear in the Project Explorer along with objects defined for any other third-party application dictionaries.

Note that the Microsoft Dynamics GP windows, reports, and fields have not all been automatically added to your VBA project. Instead, when working with VBA, add any objects you may want to use. After adding them to your project, you can reference them within your VBA code.

HINT: *The VBA Editor enables you to develop and maintain a single Microsoft Dynamics GP project at a time. To work with several different projects, store each project file in a separate location. When you need to work on a project, copy it to the same location of the Microsoft Dynamics GP application, then rename the file DYNAMICS.VBA.*

References

During the development of your VBA project, you can use objects from sources other than the current application. For example, you can access windows and fields in another third party dictionary or resources in a Microsoft® Office application such as Microsoft® Outlook®. To make these other objects available to your VBA project, add a reference to the application that contains those objects. These references are added to the project using the References window in the Visual Basic Editor. Selecting References on the Tools menu accesses this window (TOOLS→REFERENCES).

CAUTION: *Circular References are not supported in Visual Basic for Applications. Once a reference is defined between one VBA project and another, a second reference cannot be defined in the opposite direction. For example, if a reference is established from a Microsoft Dynamics GP VBA project to a third party VBA project, you cannot open the third party VBA project and create another reference back to the Microsoft Dynamics GP VBA project. Similarly, if a reference is established from a third party VBA project to the Microsoft Dynamics GP VBA project, you cannot open the Microsoft Dynamics GP VBA project and create another reference back to the third party VBA project. When you try to save the circular reference an error shows indicating that is not allowed.*

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

Methods, Properties and Events

There are a number of methods and properties defined for each type of Microsoft Dynamics GP object. These methods and properties deal with how different objects are handled. They are written using standard VBA object.method and object.property syntax. In addition, there are also events for most of the Microsoft Dynamics GP objects. Events specify when the VBA code should be executed.

Following are brief explanations for methods, properties, and events.

Methods

Methods are actions performed for a given object. There are methods available for such actions as opening and closing a window or moving a field to another position on a window. For example, the following VBA code uses the Open method to open the Customer Maintenance window.

```
'Open the Customer Maintenance window
CustomerMaintenance.Open
```

Properties

Properties are characteristics retrieved or set for a given object. There are properties available that change the title of a Microsoft Dynamics GP window or specify the position of a field. For example, the following code uses the Caption property to change the title of the Customer Maintenance window from "Customer Maintenance" to "Patient Maintenance."

```
'Change the title of a window
CustomerMaintenance.Caption = "Patient Maintenance"
```


Your use of this content is subject to your current services agreement

Events

A majority of the VBA code written is done within individual event procedures. Each event procedure executes VBA code for a specific object at a certain time, such as when the user does the following:

- Changes the value of a field
- Clicks a button
- Prints a report

For example, AfterUserChanged is an event for a field object. This event executes whenever a user changes the value of the corresponding field. In the following code, the AfterUserChanged event sets the Sort by list on the Receivables Transaction Entry window to "by Date" if the **Document Type** field is set to "Returns."

```
Private Sub DocumentType_AfterUserChanged()
  'Check to see if document type is returns
  If DocumentType = 7 Then
     'Set the sort list to "by Date"
     SortBy.Value = 3
  End If
End Sub
```

Building a User Form

User forms enable you to create windows or dialog boxes in your project and incorporate them into the user interface of the VBA host application, such as Microsoft Dynamics GP. Each form window has a **Maximize**, **Minimize**, and **Close** button. Use the buttons in the Toolbox to draw controls on the form.

User forms can also be containers for Microsoft® ActiveX® controls. Use ActiveX controls to design the elements of your user interface: buttons, lists, drop-downs, etc. When designing a form, you can add ActiveX controls to it. Then write code to make the form and the controls function.

ActiveX Controls

ActiveX Controls are components created on a user form that typically have some type of graphical representation. When the user form is in run mode, the user interacts with controls on a user form to enter data or perform an operation. These controls can be both simple and complex. For example, you could have a control that simply displays text on the user form or you could draw a complete calendar inside a box. The user can then select dates from the calendar to populate the Microsoft Dynamics GP application.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

Creating a New User Form

In the Project Explorer, select the project to which you want to add the user form. To create a user form, select User Form on the Insert menu (INSERT→USERFORM). After inserting the new user form, you can design the layout by adding controls. Also select, move, resize, or delete a control on your form.

When creating a new user form, VBA automatically marks the Microsoft Forms Object Library reference.

NOTE: *For more information on creating a User Form, refer to the help file on the Microsoft Forms Object Library.*

Setting Properties

Properties can be set for each control to better describe the object, such as the name of the user form. When creating an object, Visual Basic assigns it a default name. For example, the first command button created on a form is named CommandButton1, the second is CommandButton2, and so on. The name of the command button can be changed using the Name property. Another common property is the Caption Property. This specifies the static text that appears for some controls and is what is displayed.

HINT: *Each control has a different set of properties based on its purpose. However, many controls share a common set of properties. You can work with multiple controls by holding down the CTRL key when making your selection. You can also lasso or drag a rectangle around the controls.*

Design Mode/Run Mode

A form can be viewed in both design mode and run mode. In design mode, you can create user forms, adding controls and functionality by code you write. While creating a user form, you are already viewing the form. To view the code, right-click on the user form and select View Code. Additionally, use the buttons in Project Explorer to switch between View Object and View Code. In run mode, the form appears within the user interface, and the code runs in response to events.

Events

After designing your user form, events can be added. An event is something that happens in your application. For example, the user may click a button or change some text. If you have written code to respond to an action, it runs when the user performs that action. When an event occurs on a form, VBA looks for an event procedure to go with it.

A few common events are the Click event and the Change event. A Click event occurs when the user clicks the mouse. The Change event occurs every time you change the value of the control, such as a text box.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

The code created is placed in the Code Window. You must select the object and the event type. The name of the event is created by VBA. Then add your code in the appropriate place to perform the desired action.

Properties

User forms have properties that describe the object, such as position, size, color, and aspects of their behavior. Additional properties can be set for user forms such as the Backcolor and the Caption. Properties for user forms can be changed on the Property Sheet or through code as shown in the example below.

This example changes the Caption from "UserForm1," which was the default caption assigned by VBA, to "My User Form." The Caption is descriptive text that appears on an object to identify or describe it.

```
Private Sub UserForm_Initialize()
  UserForm1.Caption = "My User Form"
End Sub
```

Methods

In addition to properties and events, you can use methods to manipulate user forms using code. For example, there are methods available for such actions as hiding or showing an object. When an object is hidden, it is removed from the window, and its Visible property is set to False. A hidden object's controls are not accessible to the user, but they are available programmatically to the running application and to other processes that may be communicating with the application through Automation. The user cannot interact with the application until all code in the event procedure that caused the user form to be hidden has finished executing. If the user form is not loaded when the Hide method is invoked, the Hide method loads the user form but does not display it.

HINT: *If a user form is hidden, it is still available in memory and using resources. When finished with a user form, unload it to free up resources.*

The Show method displays a user form object. If the specified object is not loaded when the Show method is invoked, Visual Basic automatically loads it. When a user form is modeless, subsequent code is executed as it is encountered. When a user form is modal, the user must respond before using any other part of the application. No subsequent code is executed until the user form is hidden or unloaded. Although other forms in the application are disabled when a user form is displayed, other applications are not.

In the following example, there are two user forms used. In UserForm1's Initialize event, UserForm2 is loaded and shown. When the user clicks UserForm2, the click event is executed which hides UserForm2 and UserForm1 appears. When UserForm1 is clicked, the click event responds by showing UserForm2 again.

```
' This is the Initialize event procedure for UserForm1
Private Sub UserForm_Initialize()
  Load UserForm2
  UserForm2.Show
End Sub

' This is the Click event of UserForm2
Private Sub UserForm_Click()
  UserForm2.Hide
End Sub

' This is the click event for UserForm1
Private Sub UserForm_Click()
  UserForm2.Show
End Sub
```


Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

Variable Declarations

What is a Variable?

During the execution of program code, you often need to temporarily store a value for later use. By supplying a name for these values, you are creating a variable. The following actions can be done to the variables in your application:

- Store
- Retrieve
- Change the value

Variables are stored in memory. When using variables, you can declare them to be of a specific data type. A few examples of data types are Boolean, Integer, and Strings. VBA also supports the Variant data type, which can hold any data type if one is not specified. Even though a variant data type is large enough to hold data of any other data type, use it sparingly as it is generally slower than other variable types. Additionally, there are conversion functions, which can be used with a variant data type to change it to another data type.

Give variables meaningful names. Variable names cannot be the same as a reserved word and typically should be given names different than the field names that exist for a particular Microsoft Dynamics GP object.

Why Variable Declaration is Recommended?

Use declaration statements to name and define procedures, variables, arrays, and constants. Declaring variables results in consistency and clarity in variable names, reducing the possibility of problems within your code.

Implicit Declaration

VBA does not require you to declare variables before using them. If using variable that has not been declared, VBA creates a variant variable. This is not the recommended method.

Explicit Declaration

Declare all variables used. This makes data type intentions clear, and VBA saves time and space.

Option Explicit

To avoid the declaration problem, use the Option Explicit statement. This tells VBA that only explicit declarations are used in your modules. You cannot use a variable unless it has been explicitly declared.

***HINT** : Using the Option Explicit statement is highly recommended. It saves hours of debugging time. This option can be set by default in the Options menu item, on the Tools menu (TOOLS→OPTIONS). Mark the Require Variable Declaration option.*

Your use of this content is subject to your current services agreement

How to Declare a Variable?

Declare variables in one of two places:

- In a particular procedure (procedure-level variables)
- In the module's Declaration section (module-level variables), the area at the top of the module

There are four methods to select from when declaring a variable. Keywords indicate how long a variable keeps its value. This is typically referred to as the variables' scope.

Keyword	Functionality
Dim	The variable is available to the procedure where it is declared. It maintains its value only while the procedure is running.
Private	The variable is available to all procedures within the current module. It maintains its value while the application is running.
Public	The variable is available to any procedure within the project. It maintains its value while the application is running.
Static	The variable is available to the procedure where it is declared. It maintains its value while the application is running.

Understanding the Scope of Variables

The scope of a variable defines which parts of your code are aware of its existence. When declaring a variable within a procedure, only code within that procedure can access or change the value of that variable. However, sometimes you need to use a variable with a broader scope, such as one whose value is available to all the procedures within the same module or even to all the procedures in your entire application. Visual Basic allows you to specify the scope of a variable when declaring it.

Depending on how it is declared, a variable is scoped as either a procedure-level (local) or module-level variable.

Scope	Private	Public
Procedure-level	Variables are private to the procedure in which they appear.	Not applicable. Public variables cannot be declared within a procedure.
Module-level	Variables are private to the module in which they appear.	Variables are available to all modules.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

Variables Used Within a Procedure

Procedure-level variables are recognized only in the procedure where they are declared. These are also known as local variables. Declare them with the Dim or Static Keywords.

```
Dim strName As String
  Or
Static Accumulate as Integer
```

NOTE: *The value is reinitialized if using the Dim statement. Using the Static statement the value is retained by the variable between the calls.*

Local variables are a good choice for a temporary calculation. Multiple procedures containing the same variable name can be created. If each is declared as a local variable, it only recognizes its own version of the variable. Any one procedure can alter the value in its local variable without affecting variables in other procedures.

Variables Used Within a Module

By default, a module-level variable is available to all the procedures in that module, but not to code in other modules. Create module-level variables by declaring them with the Private keyword, as shown in the following example. When used at the module-level, the Dim statement is equivalent to the Private statement. Use the Private statement to make code easier to read and interpret.

```
Private Mymodule as String
```

Variables Used by All Modules

The values in public variables are available to all procedures in your application. To make a module-level variable available to all other modules, use the Public keyword to declare the variable, as shown in the following example. When declaring a variable as Public, refer to that variable in code using the object name.variable name. For example, VendorMaintenance.OKToDelete.

```
Public OKToDelete as Boolean
```

Module-level variables are declared in the Declarations section at the top of the module. The illustration below shows how each variable is declared.

How to Initialize a Variable

VBA assigns a default value to your variable if you do not specify a value when the variable is created. The variable has the default value until you assign it a different value.

The following list describes the initial values for some different data types:

Data Type	Initialized Value
Numeric data types	0
Strings	"" (zero-length string)
Dates	12/30/899 12:00:00 AM
Boolean	False
Variants	Empty

Compiling and Debugging VBA

Compiling

VBA enables you to compile the code written for your application. By clicking Compile Microsoft Dynamics GP on the Debug Menu, you can compile the code within the your vba file, such as the Dynamics.vba file. This ensures that your code is compiled in VBA and can then be executed within Microsoft Dynamics GP. Doing this helps you catch errors in your code prior to the code being executed.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

Upon compiling, if incorrect syntax is used, VBA gives a warning message as follows.

Debugging

VBA contains a powerful, yet easy to use debugging tool that is indispensable for the VBA developer.

Breakpoints

The first step in using the VBA debugger feature is to set a Breakpoint on a desired line of code. A breakpoint allows each line of code to be "stepped" through as it runs so that field values and other conditions can be evaluated. Generally, breakpoints are set on the first line of subroutines or on an executable line of code within the routine. Do not place breakpoints on Dim statements or on Set statements. To set breakpoints click in the gray area on the left side of the code window.

A visual indicator appears in the margin, and the line on which to break appears highlighted. Return to Microsoft Dynamics GP, and perform tasks necessary to invoke the code. When the event is triggered that causes the code to run, focus leaves the Microsoft Dynamics GP application and activates the debugger, returning to the VBA IDE at the breakpoint with a flashing cursor at the line that runs next. There is also a visual indicator on the toolbar signifying break mode.

Using F8 (or DEBUG→STEP INTO) causes each subsequent line of code to execute as it is reached.

Use SHIFT + F8 (or DEBUG→STEP OVER) to execute the next statement in the current procedure.

Use CTRL + SHIFT + F8 (or DEBUG→STEP OUT) to execute the remaining lines in the function.

Use F5 to continue the code until the next breakpoint is encountered. This is the same as clicking the **Run** button on the Toolbar in VBA.

Viewing the Values of Variables When Stopped at a Breakpoint

One method of evaluating current field values is to hold the cursor over the field in question while stepping through the code, revealing the field contents before and after the line has executed.

Immediate Window

Another way to return field values is to use the Immediate Window. After the breakpoint has been reached, open the Immediate window from the Toolbar (VIEW→IMMEDIATE WINDOW or CTRL + G). Two basic commands used in the Immediate window are Print or ?, followed by the field or variable name. Click enter to run the Print statement or query.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

The immediate window is particularly useful when your code relies on the content of a dialog box. Even though the prompt string can be evaluated by holding the cursor over the word "PromptString" in the code, comparing the string to the search string typed in your code can be difficult. The Immediate window allows you to copy the string from the immediate window and paste it into the code to assure its preciseness.

Locals Window

With the Locals window view the values, properties, and states of all of the objects on the current window (VIEW→LOCALS WINDOW). The window also displays the object type which is useful in debugging errors that allude to incompatible data types that may occur when attempting to assign the value of one field to another. The top level of the view displays the fields directly associated with the routine that caused the break to fire.

The VBA syntax "Me" is used to represent the current window. Clicking on the plus sign expands the view to display window properties as well as window objects.

Clicking on the plus sign beside a field name expands the view to display all field properties, including value.

Call Stack

The Call Stack displays the functions and/or procedures leading up to triggering the breakpoint (VIEW→CALL STACK).

Knowing which functions and procedures have been called, and in what order, may be important in routines where other subroutines or functions are called prior to the end of the current routine.

Watches

Watches can be added to track elusive problems. Enter the watch expression, field, or variable name, and select where it is to be implemented. The watch may be set up to break on an individual "module," or Microsoft Dynamics GP code window, or it may pertain to the entire project.

To make accessing the debugging tools easier, view the Debug Toolbar (VIEW→TOOLBAR, click **Debug**).

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

Modules

The Project Explorer window shows each element of your VBA Project. It includes any objects added to your project, and any new VBA forms and modules created.

The Forms folder contains User Forms created within the project.

A Module is a named unit consisting of one or more procedures, plus declarations common to all procedures in the module. VBA allows you to place all procedures in a single module, but for ease of tracking, it is recommended that you divide related procedures into separate modules. There are two types of modules available to VBA, standard modules and class modules.

- The Modules folder contains standard modules, which are the most common module. These modules can have generic procedures for code you want to run.
- The Class Modules folder contains class modules used to define custom objects, their properties, and methods. Classes define what kind of data the object can contain in its properties and how the object behaves through its methods. Classes can range from simple to complex. Class modules can be used for object-oriented programming. ActiveX then provides the mechanism to share and use objects across applications. Use Class Modules to create and store common routines that are called from multiple places within the project.

To create a Module or a Class Module, select the appropriate type from the Insert menu.

VBA Help

For more information on basic VBA topics, use the VBA Help file. Navigate to the online Help file by clicking (HELP→MICROSOFT VISUAL BASIC HELP) or by using the shortcut which is the F1 function key.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement

Conclusion

In this chapter we discussed the various components of the VBA Editor. You learned that Microsoft Dynamics GP objects must be added to the VBA project in order to access them and/or manipulate them using code. User Forms were discussed and the Properties window was shown illustrating how to change properties for objects. We talked about variables, how to declare them, and why you should declare them before using them in code.

Quick Interaction: Lessons Learned

Take a moment and write down three Key Points you have learned from this chapter:

1.

2.

3.

Microsoft Official Training Materials for Microsoft Dynamics ™
Your use of this content is subject to your current services agreement