Database Automation using VBA

UC BERKELEY EXTENSION

MICHAEL KREMER, PH.D.

E-mail: access@ucb-access.org

Web Site: www.ucb-access.org

ICON KEY	
	Note
	Example
	Warning

5th Edition

TABLE OF CONTENTS

Day 1

Section 1

1. Introduction to VBA

VBA is a programming language based on Microsoft Visual Basic(VB) language. It is a structured and procedural language. Structured programming essentially means that the programming language and its control structures determine the flow of the program. Structured programming does not use any ***Goto*** statements to force the flow of the program (= unstructured programming).

A procedural programming language specifies the ***What*** and the ***How***, meaning what is the goal and how do you want to reach this goal. Procedural programming languages are also referred to as imperative programming languages (=specifying the steps the program must take to reach the desired state). For example, SQL is a non-procedural language, you only specify the ***What***, but not the ***How***.

1.1. What is VBA?

Visual Basic is a stand-alone language that allows you to build complete applications in a windows environment. Visual Basic for Applications (VBA) is a hosted language, which means it runs within another application. VBA is most commonly used with the common office applications (Word, Excel, Powerpoint, Access), but it is also licensed to third party applications (WordPerfect, Corel Draw).

VBA interacts very easily with the host application by pointing to the object library of the host application. An object library is a file that exposes all objects of the host application to VBA. For example, in Word, an object is the Word application itself, but also a document, for instance. Being able to reference these objects gives you the power to programmatically manipulate these objects.

VBA is a subset of Visual Basic (VB). It is very similar to VB, but it does not contain all the functionalities of VB since it is hosted within another environment. The hosting environment (such as MS Word or Access) builds the foundation for using VBA.

Notes:

1.2 VBA Core Elements

VBA is comprised of some core elements that form the basic framework of the VBA environment. These core elements include objects, properties, methods, events and variables. They are explained in more detail below.

Objects

An object is generally thought of as a physical thing. For example, an “automobile” object gives you access to the entire family of cars and its objects.

VBA is an object-oriented programming (OOP) language. This kind of language enables programmers to define complete data structures from data types to the operations that can be applied to the data structure. You can create an entire object that contains both data and the operations that the object can perform. It is also possible to create relationships between these objects.

There are many objects exposed by the object library of a particular hosting environment. You can also reference other library objects, for example, you can perform a Word mail-merge from within Access using data in an Access database.

Properties

A property is a physical attribute of an object that further defines the object. Each property can have multiple values. For example, a car has the color property. The color property can have many different values, yet only a limited number of values prescribed by the automobile maker.

Some properties can be easily changed, other are more inherent to the specific object and cannot be changed. For example, it is easy to change the color of an automobile. However, a car usually only has only 4 wheels, and that cannot easily be changed.

Objects in VBA also have properties. For example, the form object contains a border style property. This property has four possible values, that is ***None***, ***Thin***, ***Sizable***, and ***Dialog***.

Methods

A method is an action that can be performed by or on an object. Again, the car example may make this easier to understand. For example, you can invoke the start method of the engine (compare this to a property), release the parking break, shift gears or turn the stirring wheel. All these actions act on the object car, and if well programmed, make the car move in the right way.

Notes:

In VBA, objects also have methods. For example, a report object can be previewed or printed. A form object can be re-queried if the underlying data has changed.

Events

An event is something that happens to an object. For example, the door closes when you pull it. Or the car turns when you turn the steering wheel. Closing and Turning are events. Events and method are closely related, it is the cause-and-effect way. An event happens when the user does something. The action of doing something is the method.

In VBA, events are used to connect programming logic to an application. Microsoft Access is based on the event-driven programming model. There is no need to write a main program that controls the flow of all the sub programs. This part is already built into MS Access, whether you use it or not. Rather than you as a programmer the user itself causes the programming logic to happen, based on the actions (=methods) of the user that cause events to happen.

For example, when you open an Access form, the ***OnOpen*** event is raised. You can use this event to write a piece of code to initialize the form, or based on a certain condition to filter data on the form. When you close the form, the ***OnClose*** event happens. You could check whether there are any pending data changes on the form and alert the user. In fact, you can also cancel certain events. We will discuss this later.

Variables

Another important of almost every programming language is the concept of a variable. Although implemented in different ways, the concept is the same. A variable is a location in memory (RAM) where you can store a value while your code is executing. Once a variable is created in memory, VBA can very easily find and remember the value than retrieving it from a database or some other place. A variable is useful if you need a particular value over and over in your code. At the beginning of your code, you would assign a value into your variable, for example a field value on a subform. Then instead of reading this value from the subform every time you need it, you simply use the variable value.

Another advantage is code readability and efficiency. What if you had hard-coded the value into your program unit? Next time you want to use that piece of code, you need a different value. Having a variable allows you to change the assignment once in your code (preferably at the beginning of your code) and then your program works like a charm since the variable name is used in your program rather than a hard-coded value.

We will discuss variables in much more detail later in this class.

Notes:

1.3 VBA Program Structure

Since MS Access is event-driven, there is no need to build a main program which controls the flow of sub programs. However, each program responding to an event is a so-called main program, which may or may not have sub programs.

A program is a set of instructions that an application can execute. A program generally is either a function procedure or a sub procedure, which in turn can include sub procedures and/or function procedures. Sub procedures and function procedures are collectively referred to as procedures. We explain later what the difference is between sub procedure and a function procedure.

The procedures are the place where VBA code resides. They act as container to hold the code. For better readability, procedures should be broken down into smaller, manageable pieces.

Now let’s discuss the difference between a sub procedure and a function procedure.

Sub Procedure

A sub procedure does not return a value. That is very important to understand, because sub procedures cannot be used within MS Access where a return value is required. For example, the Control Source property of an unbound control of a form cannot be a sub procedure because no value is assigned.

A sub procedure performs a set of tasks, for example modifying data in the database using DAO or ADO. A sub procedure can also verify data and then issue a message box to the user.

Function Procedure:

A function procedure always returns a value, even if it is the Null value. Therefore, functions can be used in expressions or in Control Sources of unbound controls, for example.

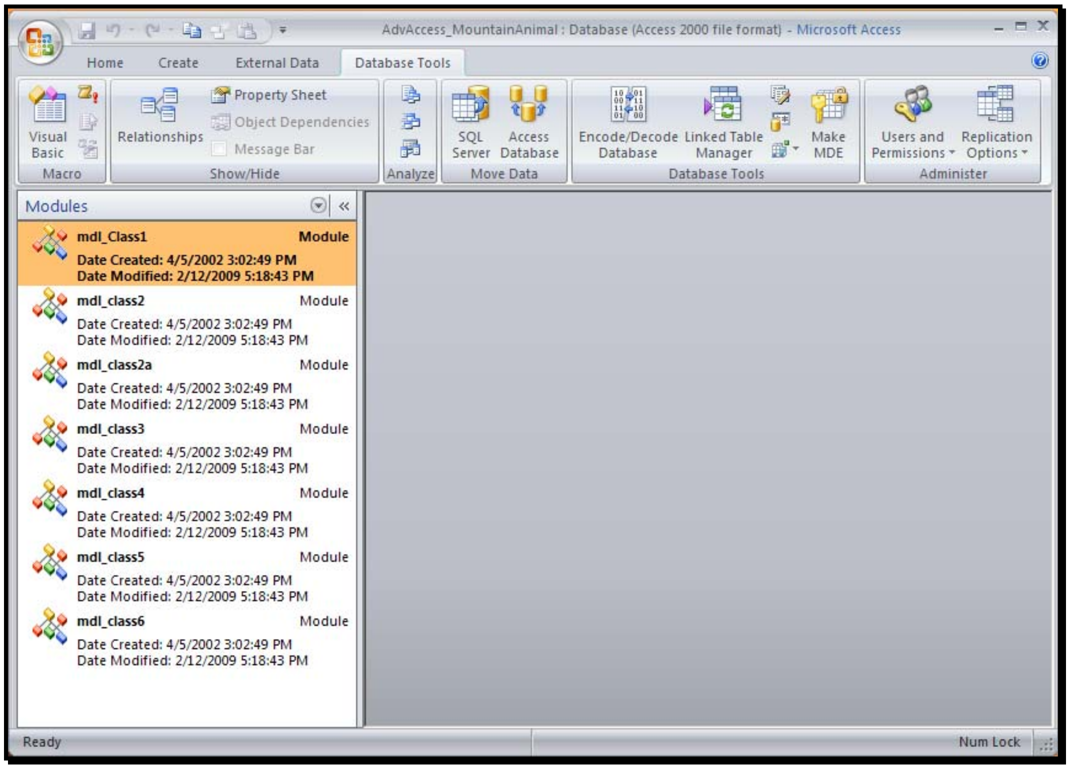
A function also performs a set of tasks. That set may be unrelated to the returning value, therefore a function procedure can also act as a sub and function procedure together.

VBA code is contained in either sub or function procedures. Many procedures are contained in the so-called code modules, the last object in the navigation pane. Furthermore, modules hold declarations, a part of a program to name variables and to set some other global parameters. In MS Access there are two kinds of modules:

Notes:

Standard Module

These are the modules displayed in the main navigation pane when the option ***Modules*** is selected in the menu of the main navigation bar. These objects are not associated with any particular object (such as form or report), and they contain sub and/or function procedures which can be called from any object.



mdl_Class1	Module	Date Created: 4/5/2002 3:02:49 PM Date Modified: 2/12/2009 5:18:43 PM
mdl_class2	Module	Date Created: 4/5/2002 3:02:49 PM Date Modified: 2/12/2009 5:18:43 PM
mdl_class2a	Module	Date Created: 4/5/2002 3:02:49 PM Date Modified: 2/12/2009 5:18:43 PM
mdl_class3	Module	Date Created: 4/5/2002 3:02:49 PM Date Modified: 2/12/2009 5:18:43 PM
mdl_class4	Module	Date Created: 4/5/2002 3:02:49 PM Date Modified: 2/12/2009 5:18:43 PM
mdl_class5	Module	Date Created: 4/5/2002 3:02:49 PM Date Modified: 2/12/2009 5:18:43 PM
mdl_class6	Module	Date Created: 4/5/2002 3:02:49 PM Date Modified: 2/12/2009 5:18:43 PM

Figure 1: Standard Modules in Access 2007

Modules contain sub procedures and function procedures, they simply act as a container to hold the procedures. You cannot run a module, therefore the Open option of the shortcut menu (right-click on the object) in the main database window for modules does not exist.

Sub procedures and function procedures should be grouped by some kind of category, and consequently procedures belonging to the same category should be placed into the same module. There are no specific rules about categorizing the procedures. However, later we will see that variables and their declaration have an impact on grouping of procedures

Notes:

Class Modules

There are three different kinds of class modules:

- Form class modules
- Report class modules
- Custom class modules

Class modules are associated with a particular object, such as forms or reports. In addition, the developer can create its own class modules. Class modules look exactly the same as the standard module, however, they are not displayed in the main database window.

Class modules are linked to their objects, and when exported, the module is exported with the object together. Forms and reports class modules contain sub and function procedures pertaining to the form or report only.

To access a class module, click on the code button in form or report design.(see figure 2) Another way of accessing class modules is through the VBA editor and its explorer window (see figure 3).

Note: We will introduce the VBA Editor in the next chapter. By then, it becomes much clearer how the project explorer works. Basically, all your VBA code of an Access application is managed using the VBA Editor.

Notes:

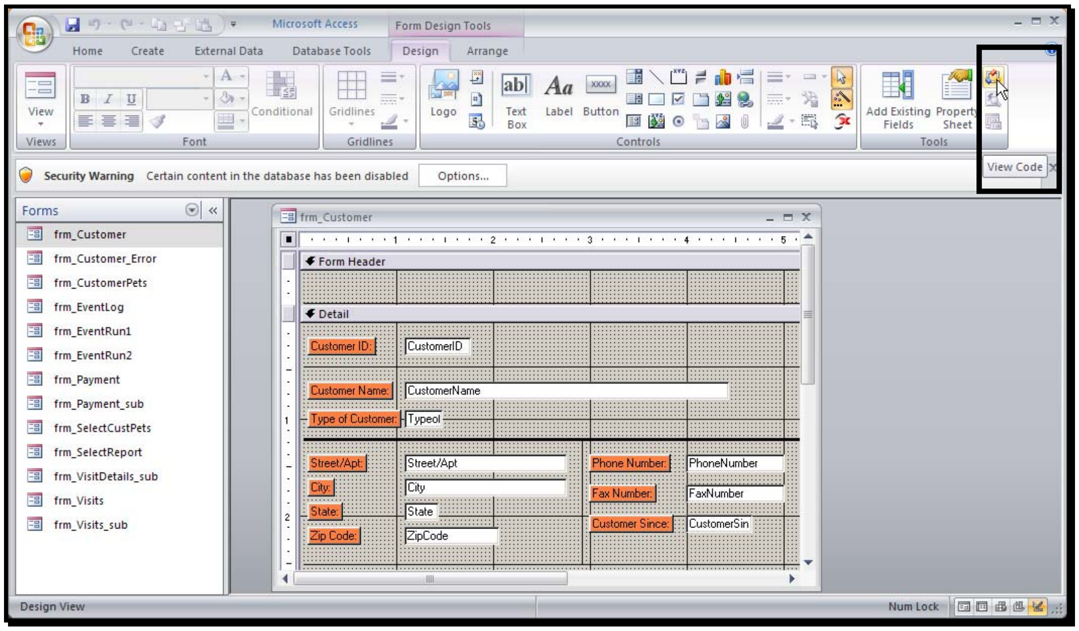


Figure 2: Accessing a Class Module

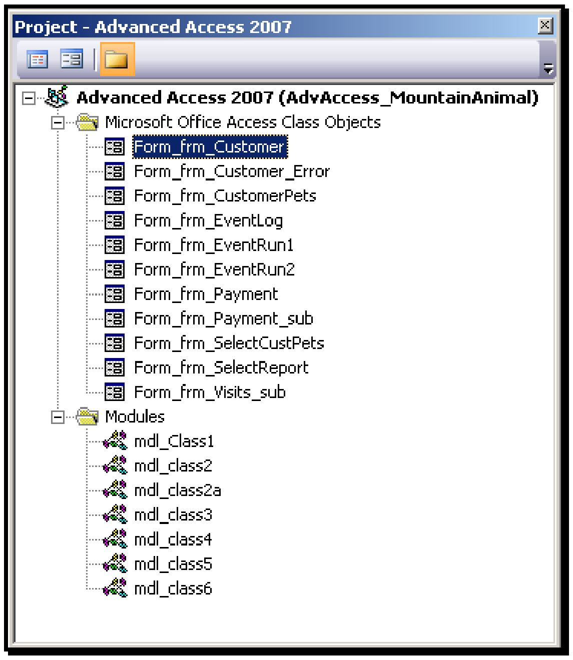


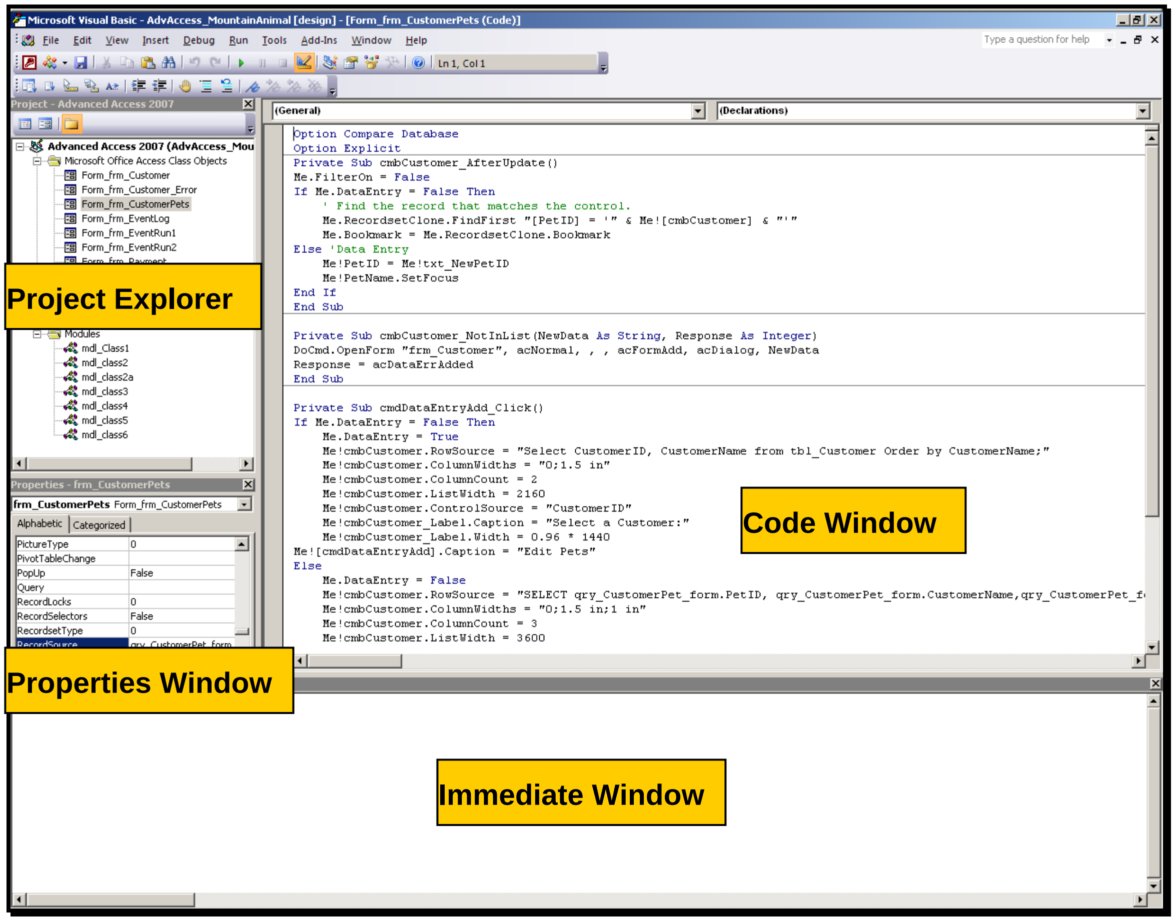
Figure 3: Accessing Class Module using Explorer

Notes:

1.4 Anatomy of the VBA Editor

As you may have already noticed, the VBA Editor is a separate application (look at the Windows taskbar), yet it is inherently connected to the Office applications. You cannot run the VBA Editor as stand-alone application (for that you need Visual Basic or Visual Studio). The VBA Editor is a powerful tool to manage all your code for your office applications. At first, it appears a bit intimidating, so let's break it down into its main building blocks and its associated functionalities.

To access the VBA Editor, simply press [Alt+F11] key, double-click on a module in the main database window or click the code (Figure 2) button when in form or report design.



PictureType	0
PivotTableChange	
PopUp	False
Query	
RecordLocks	0
RecordSelectors	False
RecordsetType	0
RecordSource	qry_CustomerPet_form

```
Option Compare Database
Option Explicit
Private Sub cmbCustomer_AfterUpdate()
Me.FilterOn = False
If Me.DataEntry = False Then
    ' Find the record that matches the control.
    Me.RecordsetClone.FindFirst "[PetID] = '" & Me![cmbCustomer] & "'"
    Me.Bookmark = Me.RecordsetClone.Bookmark
Else 'Data Entry
    Me!PetID = Me!txt_NewPetID
    Me!PetName.SetFocus
End If
End Sub

Private Sub cmbCustomer_NotInList(NewData As String, Response As Integer)
DoCmd.OpenForm "frm_Customer", acNormal, , , acFormAdd, acDialog, NewData
Response = acDataErrAdded
End Sub

Private Sub cmdDataEntryAdd_Click()
If Me.DataEntry = False Then
    Me.DataEntry = True
    Me!cmbCustomer.RowSource = "Select CustomerID, CustomerName from tbl_Customer Order by CustomerName;"
    Me!cmbCustomer.ColumnWidths = "0;1.5 in"
    Me!cmbCustomer.ColumnCount = 2
    Me!cmbCustomer.ListWidth = 2160
    Me!cmbCustomer.ControlSource = "CustomerID"
    Me!cmbCustomer_Label.Caption = "Select a Customer:"
    Me!cmbCustomer_Label.Width = 0.96 * 1440
Me![cmdDataEntryAdd].Caption = "Edit Pets"
Else
    Me.DataEntry = False
    Me!cmbCustomer.RowSource = "SELECT qry_CustomerPet_form.PetID, qry_CustomerPet_form.CustomerName,qry_CustomerPet_f
    Me!cmbCustomer.ColumnWidths = "0;1.5 in;1 in"
    Me!cmbCustomer.ColumnCount = 3
    Me!cmbCustomer.ListWidth = 3600
```

Figure 4: The VBA Editor

Notes:

Project Explorer

This window works like the windows explorer and displays all code modules in the current database. Currently, there are two main folders, class modules and standard modules. Later on, we will briefly discuss custom class modules, another type of module that is displayed in a third folder in the project tree. The project explorer enables the user to access all modules from one single point of entry.

In general, the project carries the same name as your current database. If you do not see the project explorer, simply navigate to menu View, Project Explorer or press [Ctrl+R] keys.

Properties Window

The properties window displays properties of a currently open object, such a form or report. This is the same as the property sheet in form or report design, however, it is organized a bit differently. There are two tabs, alphabetic and categorized. The different categories are Data, Events, Format, and Other, the same categories as in the property sheet. This window does not display any properties if no object is currently in design view.

To display this window, simply press the [F4] key.

Code Window

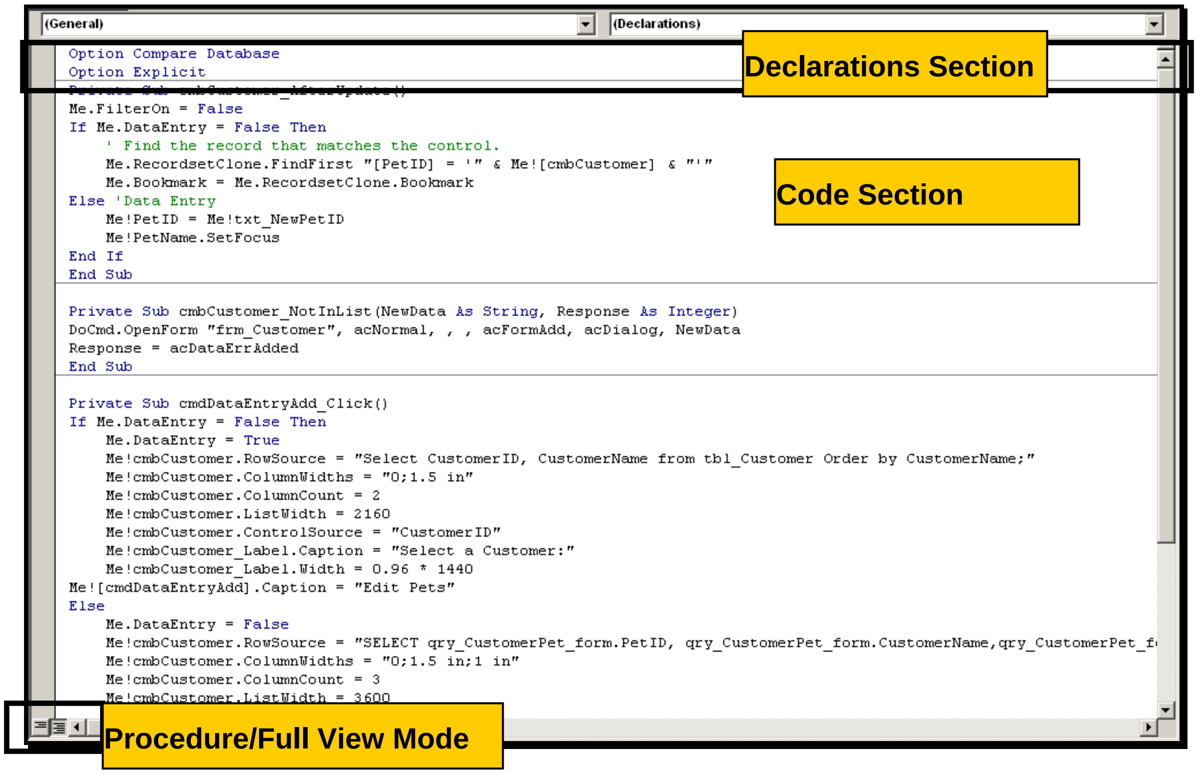
Here resides the actual code, and this is where most of this class' focus will be. The code window itself is divided into multiple sections:

- Declaration Section:
  This sections is used for variable declarations and certain other module specific options.
- Procedure Section:
  This section is reserved for sub and function procedures. Sub routines and functions are separated by horizontal lines. These lines are inserted automatically, there is no need for the user to take any additional action.

The lower left hand corner of the code window displays two buttons. When you hover your mouse over those buttons, you will see that one button is named ***Procedure View*** and the other one ***Full Module View***. This allows you to display only the currently active procedure (where the cursor resides) and to focus just on this procedure (see figure 5).

When you restore the Code Window, you will actually see all other modules behind the currently active Code Window. The VBA Editor loads the entire project into memory.

Notes:



```
Option Compare Database
Option Explicit
Private Sub cmbCustomer_AfterUpdate()
Me.FilterOn = False
If Me.DataEntry = False Then
    ' Find the record that matches the control.
    Me.RecordsetClone.FindFirst "[PetID] = '" & Me![cmbCustomer] & "'"
    Me.Bookmark = Me.RecordsetClone.Bookmark
Else 'Data Entry
    Me!PetID = Me!txt_NewPetID
    Me!PetName.SetFocus
End If
End Sub

Private Sub cmbCustomer_NotInList(NewData As String, Response As Integer)
DoCmd.OpenForm "frm_Customer", acNormal, , , acFormAdd, acDialog, NewData
Response = acDataErrAdded
End Sub

Private Sub cmdDataEntryAdd_Click()
If Me.DataEntry = False Then
    Me.DataEntry = True
    Me!cmbCustomer.RowSource = "Select CustomerID, CustomerName from tbl_Customer Order by CustomerName;"
    Me!cmbCustomer.ColumnWidths = "0;1.5 in"
    Me!cmbCustomer.ColumnCount = 2
    Me!cmbCustomer.ListWidth = 2160
    Me!cmbCustomer.ControlSource = "CustomerID"
    Me!cmbCustomer_Label.Caption = "Select a Customer:"
    Me!cmbCustomer_Label.Width = 0.96 * 1440
Me![cmdDataEntryAdd].Caption = "Edit Pets"
Else
    Me.DataEntry = False
    Me!cmbCustomer.RowSource = "SELECT qry_CustomerPet_form.PetID, qry_CustomerPet_form.CustomerName,qry_CustomerPet_f
    Me!cmbCustomer.ColumnWidths = "0;1.5 in;1 in"
    Me!cmbCustomer.ColumnCount = 3
    Me!cmbCustomer.ListWidth = 3600
```

Figure 5: Code Window and its Sections

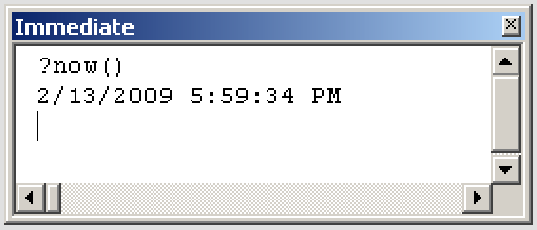
Immediate Window

The immediate window is like a sandbox; code fragments and expressions can be tested here. This is important for debugging purposes. This is a very powerful tool for exploring code and also as a place to temporarily display information from executing code. To display this window, simply press [Alt+G] keys.

Example 1-1: The Immediate Window

Simply type ?Now() in the immediate window

and press the [Enter] key.



```
?now()
2/13/2009 5:59:34 PM
```

Notes:

1.5 VBA Code vs. Macros

Macros in MS Access have a different meaning than macros in Word or Excel. Recording a macro in Word, for example, initiates a process where keystrokes and mouse actions are recorded and translated into VBA code. In MS Access, macros are step by step VBA statements, where the user has to supply arguments and parameters.

Macros are a great tool for learning VBA as well as learning object references. They can be used for simple tasks, such as opening or closing a form. Furthermore, one does not have to remember the exact syntax, the macro interface guides the user with the tasks of creating an action set.

Macros in Access 2007 have been greatly enhanced, a surprise move since the macro language in Access has not been significantly changed since Access 2.0 (1994). Therefore, you can now accomplish tasks using macros that were simply not possible in prior versions of Access.

Two of the major enhancements in macros are the use of temporary variables and embedded macros. Embedded macros allow you to combine macros and a form or report into one object. This makes it much easier managing the many objects an Access application may contain.

But still, macros do not meet the requirements for building a robust application.

Why then use Visual Basic when macros are so easy to use? There are many limitations in macros, and we will discuss the most important ones here. Macros are very good for getting started with VBA, but sooner or later, one will use only VBA to accomplish all programmatic tasks.

With Visual Basic, the developer can create their own individual functions. This functionality is not possible with macros.

When an error occurs in MS Access, usually a message box appears offering the user some kind of choices. The most often selected choice is to stop the process and to wait until the developer can identify and fix the problem. With Visual Basic, error handling can be implemented, that is, under certain circumstances the process may not stop and continue working. The user is not confronted with the unfamiliar message boxes, and this makes this process much more efficient. In addition, if an error cannot be handled at run-time, the MS Access error message can be replaced with customized error messages displaying more and useful information to the user.

With Visual Basic, objects, such as forms and reports can be manipulated at run-time. This is true for macros too, however, more controlled manipulation is possible with VBA. Also, with code, objects can be programmatically created.

Notes:

Because Visual Basic for Applications is the uniform language within MS Office, communicating with other MS Office applications is a fairly easy task. This is called Automation, formerly known as OLE (Object Linking and Embedding). Another data exchange protocol is DDE, the dynamic data exchange. Using VBA DDE can be initiated and handled. MS Access can act as a DDE client or as a DDE server.

Using VBA data modifications can be made for one record at a time. Remember that macros can only act on the entire record set by using action queries.

One of the most important features of VBA is the ability of passing values at run-time, something macros cannot do at all. This enables the developer to programmatically change the title of a form, for example, depending on certain run-time conditions.

Anther very important feature is transaction processing. This feature enables the developer to encapsulate database operations which are dependent upon each other into a workspace. By using a workspace, database operations are not saved (committed) until explicitly done so. If one of the operations fails, the entire operation can be rolled back or undone.

The Window's API (Application Programmer Interface) is a great tool to interface with the windows operation system. To incorporate windows operations into an Access application, use VBA to interface with API. For example, for a procedure which checks table links the File Open dialog box can be called to point to the location of the data file.

Replication is another powerful feature of MS Access. Replication allows multiple databases to be updated at the same time without a network connection. The master database then merges all database updates together and identifies possible data conflicts. VBA can be used to automate this process.

As you can see, there are many, many reasons to use VBA over macros. It seems that we do not need macros at all anymore. However, there are still some market niches for macros.

One feature of macros that cannot be reproduced by Visual Basic is the generation of the so-called "Hotkeys" or "Autokeys". Macros can be attached to keystroke combination, such as the [Ctrl] or the [Alt] key with any key together. For example, opening the customer form can be assigned to [Ctrl] + [C].

Note: Using the action "RunCode", even Visual Basic code can be attached to a macro, and in turn, that macro can be assigned to a Hotkey.

Notes:

<u>Note:</u> Any key combinations used by MS Access are overwritten by the key stroke combination used in Hotkey macros.

Another feature is using MS Access as a DDE server. For that the so-called autoexec macro is used. The autoexec macro is automatically run when the database is opened. Simply holding down the [Shift] key while opening the database will bypass the autoexec macro.

The autoexec macro was used for creating startup options, such as running a specific code or opening a specific form. This is now replaced by the Startup option dialog box, where all these options can be set.

<u>Note:</u> In Access 2007, some action commands from older version are no longer available (for example DoMenuItem). When converting from an older version, you must edit macros container actions no longer supported and figure out a way to use a new action.

Notes:

2. VBA Basics

In this chapter we will start writing our first code samples. The concept of a variable is introduced as well as the syntax of the basic commands. Furthermore, we will learn how to name the variables and the procedures.

2.1 Basics of Variables

As we have already mentioned earlier, a variable is a location in memory where you store information that you want your VBA code to access quickly. Using variables makes your code very dynamic, efficient, and readable. We will now start writing our first procedure and learn how to use a variable.

Usage of Variables

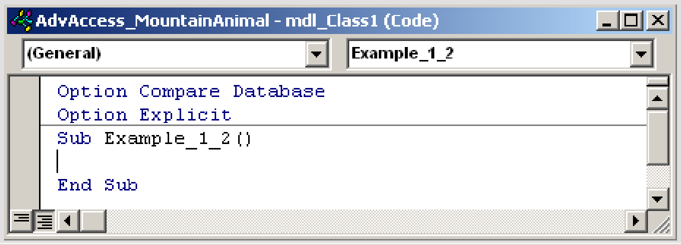
Example 1-2: The first program

1. Double-click on mdl_class1 in the main database window.

2. Now you are in the VBA Editor, simply type below the Option Explicit line:
   Sub Example_1_2

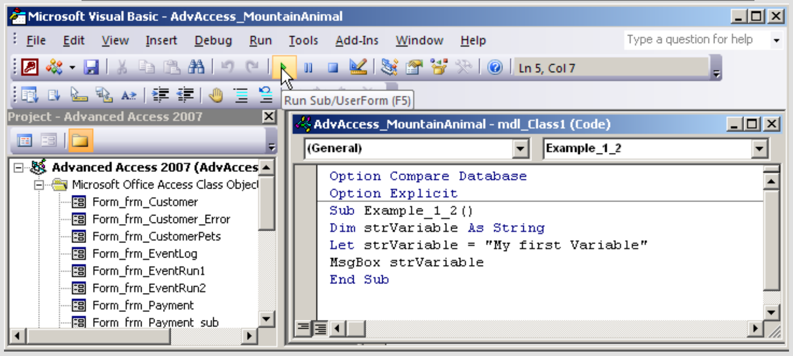
3. Press the [Enter] key and observe what happened.

4. With the cursor placed between the Sub and the End Sub, type the following:



```
Option Compare Database
Option Explicit
Sub Example_1_2()

End Sub
```


Microsoft Visual Basic - AdvAccess_MountainAnimal
File Edit View Insert Debug Run Tools Add-Ins Window Help
Type a question for help
Ln 5, Col 7
Run Sub/UserForm (F5)
Project - Advanced Access 2007
Advanced Access 2007 (AdvAcces
Microsoft Office Access Class Object
Form_frm_Customer
Form_frm_Customer_Error
Form_frm_CustomerPets
Form_frm_EventLog
Form_frm_EventRun1
Form_frm_EventRun2
Form_frm_Payment
Form_frm_Payment_sub
AdvAccess_MountainAnimal - mdl_Class1 (Code)
(General)
Example_1_2

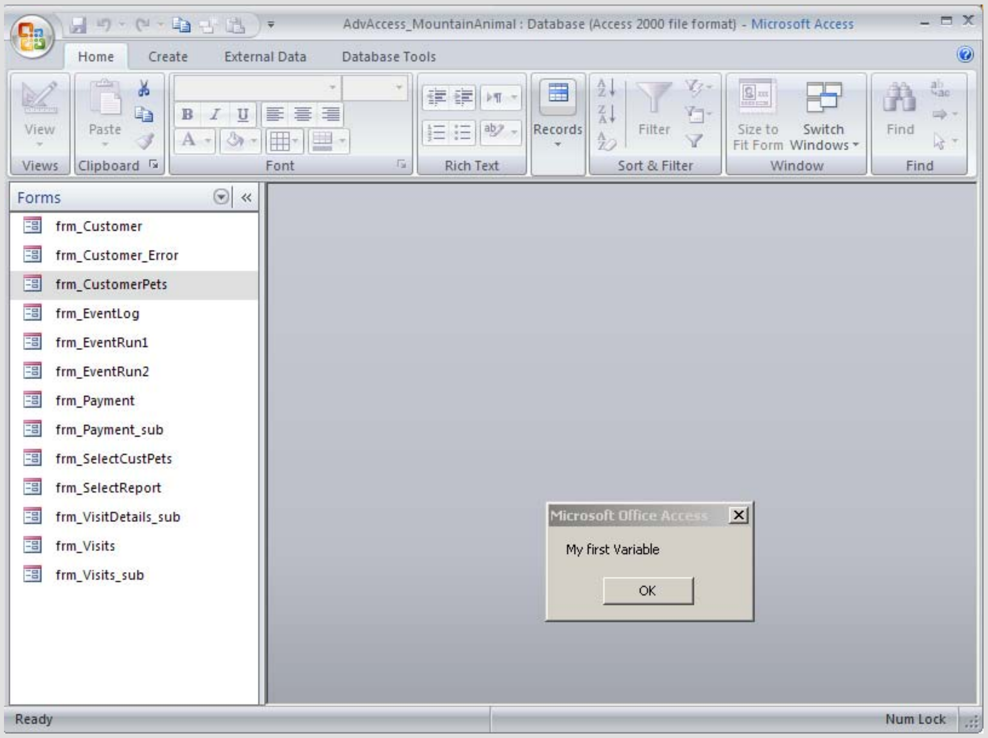
```
Option Compare Database
Option Explicit
Sub Example_1_2()
Dim strVariable As String
Let strVariable = "My first Variable"
MsgBox strVariable
End Sub
```

Notes:

Example 1-2 (continued): The first program

5. Click on the ***Run*** button (shown in the screen shot) in the toolbar

6. Now you see a message box displayed (in Access environment) showing the content of the variable:



My first Variable

7. Click on the OK button, and control returns back to the VBA Editor. The procedure is now finished running.

Warning: If you receive an error message when trying to run the code example saying that macros are disabled, do the following:

1. Navigate to the Office button
2. Click on Access Options at the bottom
3. Click on Trust Center on the left, then on Trust Center Settings
4. Now click on Macro Settings, and click on Enable all Macros
5. You must close and restart Access 2007

Notes: