Introduction to Microsoft Access and Visual Basic for Applications

Introduction to Databases and Microsoft Access

A database is a collection of information stored somewhere for easy organization and exploration. Databases are used in various situations such as keeping a record of friends, registering employees job attendance, counting the number of students in a classroom and keeping a record of their grades. This means that we have been using databases all our life.

We are going to learn how to use the computer to create and exploit databases. We will use Microsoft Access. Learning how to use Microsoft Access and some programming is just a matter of applying your already tremendous knowledge on information storage and retrieval.

Microsoft Access is a software product used for desktop database applications. You can use it for personal stuff, for home, or for your business/company.

We are going to learn not only how to create databases in Microsoft Access, but also add some programming to further tell the computer what to do with data we provide.

To further enhance the functionality of our databases in Microsoft Access, we will learn Visual Basic for Applications, a programming language that ships with Microsoft Access.

Launching Microsoft Access

You start Microsoft Access like any of the usual windows applications. The most common way consists of clicking Start -> Programs -> Microsoft Access. As a member of the Microsoft Office suite, you can start the application by clicking Start -> New Office Document. You can also create a shortcut on your desktop. Then, you can start creating your database.

Microsoft Access allows you to create a database from one of two processes: from scratch or by using one of the templates. However you start, you can add objects or fields almost as if you were switching from one technique to another.

Microsoft Access is equipped with wizards, everywhere, some of which are not even installed by default during program setup. Although wizards are mostly efficient, you should know what they are doing because more than once you will want to customize a behavior they have configured. A classic example I know is that whenever you ask Microsoft Access to insert a US ZIP Code in a table, the field is configured for a 5+4 format; most of the times, you don't need the last four digits, so you have to delete them manually. I encourage you to use wizards as long as you know what they are doing. If you want more control, you might have to do (some) things manually.

Practical Learning: Starting Microsoft Access

1. To start Microsoft Access, click Start-> Programs -> Microsoft Access.

Microsoft Access	×
Create a new database using	
🗐 🔿 Blank Access database	
Access database wizards, pages, and projects	
Open an existing file	
More Files	
C:\Programs\\College Park Auto Shop VBA1	
C:\Programs\\College Park Auto Shop V2.2	
C:\Programs\Access\College Park Auto Shop V2	
OK Cancel	

2. When Microsoft Access starts, you receive the first dialog. This dialog allows you to decide on what you are planning to do. The first radio button allows you to create a database from scratch. The second will help you create a database from one of the available and installed templates/samples. If you already have a database file on your computer, floppy disk, on someone else's computer you have access to (through a network), use the third radio button to locate it.

For this lesson, click the Open An Existing File radio button.

- Locate the VBAccess Exercises folder where you installed the exercises for this tutorial. Then locate the VBAccess Exercises folder
- 4. Click the Grier Summer Camp1 database and click Open.

Presenting Microsoft Access

Microsoft Access shares part of the look of Microsoft Office applications.

On top of the application, you see a long bar. On its left is the System icon with the usual key symbol. On the right side of the title bar, you see the windows system buttons that allow you to minimize, maximize, restore, or close the application.

Under the title bar is the Menu Bar with words like File, Edit, View, etc.

Under the menu is the toolbar. Toolbars change more regularly here than on most classic Microsoft Office applications. For example, the current toolbar is called the Form View toolbar. If you touch almost any object on your interface, a new toolbar comes up (you will have fun with them).

The most usual object of the interface is the Database Window.

• To access the Database Window, press F11.

The Database Window is made of different parts. On its own title bar, you see the name of the current database. Under the title bar is the Database Window's toolbar, this one also changes a lot depending on the object selected.

On the left side of the Database, Microsoft Access Objects are listed by categories. Once you click a category, objects of that category display on the right, wide section of the Database Window.

To do its work, Microsoft Access organizes a database into objects that we should get familiar with.

Tables

- 1. On the Database Window, click Tables. In the right section, double-click tblEmployees. The tblEmployees table opens.
- 2. After viewing the table, to close it, click the close button on its top right corner <u>×</u>.

Queries

Although looking like a table, a query is a question you ask the database about available data, and the computer responds by display appropriate results. When creating a query, you tell the database what data you are concerned with and what categories you want the database to isolate. You can use a query to define new fields based on existing ones. Therefore, there are various kinds of queries.

You create a query by defining criteria, then Microsoft Access examines available data, if it finds data that matches your criteria, it shows you the result.

From our database, we are going to create a query where we will ask the database to provide a list of employees that includes only the employee number, the last name, title, and the office location.

- 1. To open a query, click the Queries button.
- 2. Double-click **qryMembersByTShirtSize**
- 3. After viewing the query, to close it, click its Close button \blacksquare .

Forms

A form is a user friendly interface you provide to your user(s) for easier data entry. Depending on your creative and artistic ability, a form's interface is usually better looking that anything else in your database.

To create a form, you first decide where its data will come from. Since we have already seen that data in a database resided on tables, data on a form first originates from a table, it could also come from a query, from more than one table or various queries. Data on a form can also be made of calculated or combined fields. All these could be done from known fields or from programmatically created fields.

During design, you should make sure you get a fairly good looking form by using appropriate font, sizes, colors and special styles. Most of these features can also be controlled programmatically where you will ask the computer to assign particular settings such as colors, etc, when a certain condition is met.

- 1. To see a form, click the Forms button.
- 2. Double-click frmCamps.
- 3. After viewing the form, close it.

Reports

A report is the desired form of data you want to print. In other words, it is a way of telling the computer, "Based on available data, this is what I would like you to print, and this is how I want it printed.

When creating a report, you have to be a little artistic because your business might depend on it. During design, you can specify font, color, size and other special items or features. When programming, you can ask the computer to make some decisions and print conditionally on some criteria you set. For example, you can ask the computer to print student grades in grades in red if these grades are below a set minimum. The computer can also be directed to automatically print account statements on a particular date of the month, once the system calendar shows the date and time, the computer will print and get the bank statement ready for shipping.

1. To view a report, click the Reports button under Objects.

Lesson 1: Introduction to MS Access and VBA

- 2. Double-click **rptCamps**
- 3. After viewing the report, close it.

Data Access Pages

New to Microsoft Access 2000, Data Access Pages allow you to publish your information on the Internet or on an Intranet, and then share data with others.

So far, I have not seen any advantage of Data Access Pages. Therefore, for this tutorial, we will not learn and will not use them. If you want to publish your database to the Internet or to an intranet, I suggest you use ASP (Active Server Pages) or aspx.

Macros

A macro is an action or a set of actions you ask the computer to perform is response to something.

I will not encourage you to use macros, if you "inherit" legacy code from a database created by someone else or that you got from someone else, you can convert them to VBA.

Programming in Microsoft Visual Basic is not as complicated as you would think it is, at least as compared to C++. In this tutorial, I will try to be as much detailed as I can; therefore,

I WILL NOT USE MACROS FOR THIS VBA TUTORIALS

Modules

Modules are pieces of Visual Basic code that you use during programming to tell the computer what to do when, how, and using what.

- 1. To take a look at a module, click the Modules button in the Objects list.
- 2. Double-click modUtilities
- 3. View the code when Visual Basic launches.

Introduction to Microsoft Visual Basic

Although I am using both MS Access 97 and MS Access 2000 for this tutorial, the following description is given for MS Access 2000. Although the following description and screenshots are given for MS Access 2000, if you are using MS Access 97, don't worry and simply ignore it. You don't need to upgrade.



To ke most toolbars you are familiar with, to know what a particular button is used for, position your mouse on top and view a tooltip.

Almost any section of Visual Basic is dockable, which means it can be moved on the screen to another location.

The Project Explorer displays the coding objects available for your database. This window object is usually on the left of the window. To move it, click on its blue bar under the Standard toolbar and hold your mouse while you are dragging to the desired location. To position it back to its previous location, double-click its title bar.

To expand or collapse the folders tree, click the Toggle Folders button.

If the Project Explorer is not displaying, on the

main menu, click View -> Project Explorer.



Properties - Global Code 🛛 💌				
Global Cod	e Module	•		
Alphabetic	Categorized			
(Name) Glo	bal Code			
<u> </u>				

Every object and item of your database has properties associated with it. You control those properties when you design the object; for example, when you are designing a form in Microsoft Access, you define what its caption will be. You can as well have access to these properties in Microsoft Visual Basic. These properties are displayed in the Properties Window when the object is selected.

Since the Project Explorer and the Properties Window usually share the same section of the window (it is better that way), you can shrink one and heighten the other. To do that, position the mouse on the gray bar between both window. When the mouse pointer turns into a short line with double arrow, click and drag.

The Code window is the area where you will mostly be working, this is largest section of the studio. It is mainly made of three sections.

On top, there are two combo boxes. The Object combo box allows you to select a particular object and access its events, actions that the object can launch. The Procedure combo box allows you to select an action, related to the object in the Object combo box, that you want to control.

The big and wide area is where you will be writing code. There are one vertical and one horizontal scroll bars that allow you to move left and right, up and down in case your code is using more space than the code window can display.

There are two small button on the left side of the horizontal scroll bar. The Full Module View button is used to display the whole associated with an object. The Procedure View button will display only the public procedures associated with the database.

Closing Microsoft Access and Visual Basic

The version of Microsoft Visual Basic we are using here is "For Applications". Indeed, you can create a fairly functional application with this version, but it is related to Microsoft Access (or Microsoft Office). To launch it, you should be in Microsoft Access. On the main menu (of any Microsoft Office application), you would click Tools -> Macro -> Visual Basic Editor.

When you are in the editor window of Microsoft Visual Basic, you can get back to Microsoft Access either from the View Microsoft Access button on the Standard toolbar, or by clicking a Microsoft Access object on the Taskbar. The shortcut to get back to Microsoft Access is Alt + F11.

You can close Microsoft Visual Basic any time and keep Microsoft Access running. If you close Microsoft Access, Microsoft Visual Basic will be closed also.

- 1. On the Standard toolbar of Microsoft Visual Basic, click the View Microsoft Access button to get back to the database.
- 2. To close Microsoft Access, press Alt + F4.

<u>FunctionX</u>		Home	Next
	Copyright ©	2002-2004 FunctionX, Inc.	
	FUNCTIONX	MS Access Toturial	



Introduction to Message Boxes

In order for a user to interact with the machine, computer programmers equip their applications with objects called controls. Almost every type of object that displays a physical presence on your monitor screen is referred to as a control <u>as defined on</u> this site.

The most fundamental control in the world of Visual Basic is a form. A form is rectangular object that is used to hold other controls. These other controls cannot exist by themselves: they need a host or parent, which is the role of the form.

The most commonly used form or dialog box is called a message box. It is used to display a message to the user who must click a button on the message box in order to close the dialog box. A message box looks as follows:

MsgBox "Message To Display"

The word MsgBox, which stands for Message Box, is required. The message to display can be typed inside of double-quotes. At this time, that form of the simplest message box is the one we will be using until we learn more details about message boxes.

VPractical Learning: Using a Form

- 1. Start Microsoft Access. On the opening dialog box, click the first radio button to create a blank database and click OK
- 2. Change the file name to Fundamentals and click Create
- 3. On the main menu, click Insert -> Form
- 4. On the New Form dialog box, click Design View and click OK
- 5. To save the form, on the Form Design toolbar, click the Save button.
- 6. Type **frmMain** and press Enter

Introduction to Events

Microsoft Windows is an event-driven operating system. Whenever you do something on the computer such as typing, moving or clicking the mouse, the application you are using, for example a word processor, a calculator, or a personal information manager (PIM), sends a message to the operating system, which is Microsoft Windows. The action of sending a message to the operating system is called an event. When an event is sent to the operating system, the object that sends the event is said to "fire" the event. Once the operating system has received an event, it analyses what you did, interprets it, and sends you the result to the best of its interpretation. To be an effective database developer or programmer, you should be aware of events and how they work.

To use a program, you have to "Load" it into memory; the computer will do it for you. And to load a program, you have to select and start it. That's why you need to find it and...

When a program starts, it is said to be launched. Visual Basic considers that the program is Opening. It takes just a few seconds for a program to launch or open. After the program has been launched, it is said to be Loaded. Loading and running would mean the same thing, especially in Visual Basic. In reality, there are many events that the same control can send. To better manage the computer resources, the events are sent in sequence but one at a time.

These actions are events you create. There are other actions the computer performs behind the scenes, sometimes for its own maintenance. But, regardless, an event is an event. The beauty of Visual Basic programming is that you have to take care of only your initiated events, the computer takes care of itself.

VPractical Learning: Launching, Loading, And Running a Program

- 1. To start WordPad, click Start -> Programs -> Accessories -> WordPad.
- 2. While the program is Opening, you should see a brief splash screen (since WordPad is a small application (not by programming standard, but as far as the users are concerned), the splash screen might not appear, or it would be very brief):

WordPad Version 1.0 Microsoft Foundation Classes version 4.0

Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

3. Notice that, after launching, WordPad is opened. Visual Basic considers that WordPad is Loaded:

Document - WordPad	×
File Edit View Insert Format Help	
Arial 🔽 10 🔽 Western 🔽 B 🗾 🖳 🚊 🗮 🗮	
<u>.</u>	•
For Help, press F1	:

4. Return to Microsoft Access

Event Proprietorship

An event is initiated by an entity; this could be the computer, an application, a form, a control, a report, a web page, or something else. The component that initiates the event needs to control its behavior. This allows other components to trust it to handle "its own business" so the others can expect a reliable result. To accomplish that, an event is considered a private matter.

Therefore, the coding of each event starts with the Private keyword. This is to let other components know that, "Let me handle this, OK? Trust me and just see what I can do." Not all actions are private, as we will learn. But at this time, we consider that an event starts with:

Private

An event is really an assignment you ask the application, the form, or the control to perform in response to a particular action happening. You can even ask a control to perform an action based on the behavior of another control or based on the computer doing something (such as displaying 12:00 PM). The actions are called Procedures. There are two kinds of procedures: **Functions** and **Sub** Procedures. Both are written in Visual Basic.

A Function is a general assignment you write in Visual Basic. This assignment is a resource for other events or actions to get results. For example, if many controls on a form would require a particular value or the result of a particular calculation, you

can write a function that all desired events can refer to and get the appropriate result. Since other events and functions would expect a particular result from it, a function is expected to **Return** a value. And we will learn what kind of value this could be.

A **Sub** procedure is a type of assignment that applies to an event associated with a particular database, form, report, or other control. It is used to "enclose" the coded assignment you want an event to carry. For example, if you want to display a message when a form loads, you would write the code of the Load event in a **Sub** Procedure. Since each event is a procedure, now we have:

Private Sub

Each object, such as a form, in your program has a name, and we will learn that each control (buttons, combo boxes, check boxes) has a name. Each form or each control has its own events. We will also see that different controls can have the same kind of event. You ought to let Visual Basic know whose action, I mean sub procedure, you are writing the event for. This is done by specifying its name. Here is an example:

Private Sub ControlName

There are many events associated with a form or a particular control. Therefore, we need to specify what particular event we are writing code for. The name of the event is written after the name of the control. To distinguish between a control's name and its event, Visual Basic uses a convention of displaying an underscore between them, like this:

Private Sub ControlName_Event

Since a sub procedure is an assignment and there could be various assignments in your program, such an event starts with Private Sub. Now, you should let Visual Basic know where an assignment ends. this is done with **End Sub**, like this:

Private Sub ControlName_Event

End Sub

As mentioned already, and we will learn how a function returns a value, there are two important matters with the assignment an event is supposed to carry: the subject of the assignment and the necessary accessories to carry the assignment.

The subject of the assignment is called the body. It is written between the Private Sub and End Sub lines: this is where you specify what the event is supposed to accomplish. Some events just need to know what you want them to do, for example, you can ask a button to close a form when that button is clicked. This could be as simple as that. On the other hand, when the user clicks somewhere on a form, you could ask the form to display something.

Some events have or use just the body of the code to carry their assignment. Some other events will need some values from you. In some situations it will be one value, in some others it could be more. The group of values that an event needs is called argument. Again, depending on the event, this could be one argument, or it could be as many as necessary.

The argument or group of arguments that the event might need would be listed in parentheses on the right side of the Event name, like this:

Private Sub ControlName_Event(Argument1, Argument2, Argumentn)

End Sub

Even if an event doesn't need an argument, you should provide empty parentheses, like this:

Private Sub ControlName_Event()

End Sub

A form or a control has a usual event that it performs, if nothing else. The event is called the **Default** event.

Programming Events

There are various ways you can initiate an event on a control of your database application. In Microsoft Access, if you open a form or a report, you should access its Properties window. The events are usually stored on a form or a report. Therefore, you should first open the form or the report whose event you want to program, in Design View. If the form or report "carries" the control whose event you want to program, you should also first open the form or report in Design View.

- First Technique
 - 1. If you want to write code that relates to the whole form or event, double-

click its selection button which is at the intersection of the rulers . If you want to write code for a control, on the form or report, double-click the control to display its Properties window.

2. Once the Properties window is displaying, you have a lot of alternatives. Click the Event property sheet and inspect the list of events:

🖆 Form	×
Format Data Event Other All	
On Current	
Before Insert	
After Insert	
Before Update	
After Update	
On Dirty	
On Delete	
Before Del Confirm	
After Del Confirm	
On Open	
On Load	
On Resize	
On Linload	<u> </u>

- 3. After locating the event you want, double-click it. The [Event Procedure] clause will be added to the event. Once the [Event procedure] is displaying, click the ellipsis button in to launch Microsoft Visual Basic. The keyboard caret would be positioned in the event and wait for you.
- Second Technique

- 1. On the Properties window, click the event to reveal its combo box.
- 2. Click the arrow of the combo box and click [Event procedure].
- Third Technique
 - 1. Right-click the form, report or control whose event you want to write code for.
 - 2. Click Build Event...



- 3. On the Choose Builder dialog box, click Code Builder and click OK. This would launch Microsoft Visual Basic with the default event of the form, report, or control
- Fourth Technique
 - 1. As we saw earlier when reviewing the events, first open a form or report in Microsoft Access.
 - 2. To launch Microsoft Visual Basic, click the Code button in Microsoft Access.
 - 3. In the Object combo box, select the form (or report) to launch its default event.
 - 4. In the Procedure combo box, select the event you want.

Sometimes you or Microsoft Visual Basic will have inserted an event that you didn't want or that you don't want to program. If this happens, simply ignore the event: you do not have to delete it because if an event has been initiated but no code was written for it, Microsoft Visual Basic will take care of deleting it when you save the form or report.

Anatomy of an Event

The above introduction was meant to show you what a coded event looks like. Microsoft Visual Basic will do a lot of work for you behind the scenes. For example, it will always set a beginning and end event for you, reducing your headache. It will also specify the names (and types) of arguments for you.

Your programming will occur in Microsoft Visual Basic. Whenever the behavior you want to implement cannot be achieved in Microsoft Access, you will have to launch MS VB.

Practical Learning: Launching, Loading, And Running A Program

- 1. In Microsoft Access, make sure the frmMain form is opened in Design View
- 2. On the Database toolbar, click the Code button
- 3. Notice that the Code Editor displays
- 4. Type:

Private Sub Form_Load

- 5. Press Enter.
- 6. Notice that Visual Basic added the parentheses and the End Sub line.
- 7. Also notice that the cursor is positioned between the Private line and the End Sub line.

Form's Events

Every Windows control you will use in your application has a set of events it can fire to carry its assignments. This is also valid for a form. Some of the events, as you will see, are appropriate for a particular control or an action performed on the database. Some and many other events are shared by various controls. Nevertheless, as a host for other controls, a form has specific events it can fire.

The Open Event

The OnOpen event is the first event a form fires when it is opened or when it is switched from Design View to Form View. You can use this event to perform some checking on the form or its controls while the form is opening but before it formally displays to the user

The Load Event

After the form opens, it installs itself in the computer memory. When installing itself, the form fires an OnLoad event. This is the favorite event you can use to perform any action or operation at startup. This event fires before the user can do anything but when the operating system is aware of the form. Once the form has been loaded, it becomes a regular control but can still manage the presence of other

Lesson 02: Overview of Windows Events

controls on it.

The Resize Event

For the operating system to be aware of a form as a Windows control, the form needs to be drawn on the screen (the monitor). This is because the operating needs to know the dimensions of the form. Whenever the dimensions of the form are set or reset, an event called OnResize is sent to the operating system.

The UnLoad Event

After using the form, if you close it, the memory that the form was used must be emptied and all cleaning related to other controls hosted by the form must take place. To take care of this, an event called called OnUnLoad is sent to the operating system. This is the opposite event to the OnLoad.

The Close Event

After the form has been removed from the screen, that is, once the form has been closed, an OnClose event fires.

Practical Learning: Using Form's Events

1. Type:

```
Private Sub Form_Load()
MsgBox "The form has been loaded"
End Sub
```

- 2. Return to Microsoft Access and switch the form to Form View. Notice that a message box displays
- 3. Return to the Code Editor or Visual Basic. Notice that the Object combo box is displaying the word Form.
- 4. Click the arrow of the Object combo box to display its list. The Object combo box displays a list including the form and its controls, if any is positioned on the form.
- 5. Notice that the Procedure combo box is displaying Load.
- 6. Click the arrow of the Procedure combo box to display the events of the form:

Lesson 02: Overview of Windows Events

- - 	_ 🗆 🛛	
Fo	rm 💌	Load
	Option Compare Database Option Explicit	KeyDown KeyPress KeyUp
	Private Sub Form_Load()	Load LostFocus
	MsgBox "The form has End Sub	MouseDown MouseMove MouseUp
		Open 🔒 🗐
		Timer Unload

- 7. Click Open
- 8. Change the code of the event as follows:

```
Private Sub Form_Open(Cancel As Integer)
MsgBox "The form is not opened"
End Sub
```

- 9. Switch to Microsoft Access.
- 10. Notice that the message box for the Open event displays first. After you click OK, the message box for the Load event displays. This indicates that the Open event takes precedence over the Load event.

Fundamental Events: Click

To interact with the computer, one of the most usually actions the user performs is to click the mouse. The mouse is equipped with two buttons. The most clicked button is the left one.

Because the action simply consists of clicking, when the user presses this button, a simple event, called Click is sent or fired. When the user presses the (left) button on the mouse, the mouse pointer is usually on a Windows control. Based on this, the control that is clicked "owns" the event and must manage it. Therefore, no detailed information is provided as part of the event. The operating system believes that the control that fired the event knows what to do and should take care of everything. For this reason, whenever you decide to code an OnClick event, you should make sure you know what control sent or fired the event. This is (one of) the most common events of Windows controls.

Practical Learning: Firing an OnClick Event

- 1. To see what the OnClick event looks like, on the Object combo box, select Detail.
- 2. Notice that the skeleton for the Click event of the Detail section of the form is displaying. Also notice the OnClick event does not carry much information.
- 3. Implement the event as follows:

```
Private Sub Detail_Click()
MsgBox "The form has been clicked"
End Sub
```

- 4. Switch back to Microsoft Access. Switch the form to Form View and click in the middle of the form to see a message box.
- 5. Switch the form back to Design View

Fundamental Events: The Double-Click Event

To make the mouse more useful and to verify its role, another action can be performed on it. This consists of pressing the left button very fast twice. This action initiates an event known as OnDblClick.

By default, the DblClick event provides a simple structure to the programmer. You must know what control has been double-clicked and what you want to do. This means that the operating system relies on the control that "owns" the event.

Practical Learning: Firing an OnDblClick Event

- 1. While Detail is still selected in the Object combo box, in the Procedure combo box, select DblClick
- 2. Notice the structure of the event
- 3. Remove or delete the MsgBox line of the Detail_Click event and implement the Detail_DblClick event as follows:

```
Private Sub Detail_DblClick(Cancel As Integer)
MsgBox "The form has been double-clicked"
End Sub
```

- 4. Switch to Microsoft Access and switch the form to Form View
- 5. Double-click in the middle of the form
- 6. After reading the message box, clicking OK

7. Switch to the Code Editor and delete all events or at least their contents (between the Private and End Sub lines)

Fundamentals of Coding: The Indentation

Indentation if a technique that allows you to write easily readable code. It consists of visually showing the beginning and end of an event (or procedure, as we will learn). Indentation consists of moving your code to the right side so that the only line to the extreme left are those of the Private and End Sub lines.

The easiest and most common way to apply indentation consists of pressing Tab before typing your code. By default, one indentation, done when pressing Tab, corresponds to 4 characters. This is controlled by the Module property sheet of the Options dialog box. To change it, on the main menu of Microsoft Access 97, click Tools -> Options and click the Module tab:

View General Hyperlinks/HTML Edit/Find Keyboard Datasheet Tables/Queries Forms/Reports Module Advance Code Colors	tions		det rota de tablecte tota		?
Code Colors Text Area: Normal Text Foreground: Gating Options Foreground: Auto Indent Auto Syntax Check Require Variable Declaration Compile on Demand Background: Automatic Indicator: Automatic Indicator: Sample Vindow Settings Vindow Settings Full Module View Procedure Separator Vindow Settings	View Datasheet	General Tables/Queries	Hyperlinks/HTML Forms/Reports	Edit/Find Module	Keyboard Advanced
Debug Window on Top Margin Indicator Bar	Code Colors Text Area: Normal Text Foreground: Background: Indicator: Font: Courier New Sample AaB	Automatic Automatic Siz	Coding Au Au Au Au Coding Au Au Au Au Au Au Au Au Au Au	options to Indent to Syntax Check quire Variable De- provide on Demano to List Members to Quick Info to Data Tips dth: 4 w Settings Il Module View ocedure Separationag- and-Drop Tex soug Window on Cargin Indicator Ba	or t Editing Top

If you are using Microsoft Access 2000 or 2002, or on the main menu of Visual Basic, click Tools -> Options and click the Editor property sheet.

If you don't want the pressing of Tab to be equivalent to 4 characters, change the value of the Tab Width text box to a reasonable integer value and click OK. Otherwise, it is (strongly) suggested that you keep to its default of 4 characters.

Practical Learning: Firing an OnDblClick Event

- 1. Click anywhere on the line that has MsgBox and press Home
- 2. Press Tab to change the Detail_DblClick event as follows:

```
Private Sub Detail_DblClick(Cancel As Integer)
    MsgBox "The form has been double-clicked"
End Sub
```

3. Save the form and close Microsoft Access.

Introduction to Controls Events

The controls used on Microsoft Windows applications provide the main means of interaction with the computer. To make such interaction effective, they send regular message to the operating system. There are many controls and they send various messages. Although you might have seen a great quantity of controls on some other applications or programming environments, Microsoft Access by default presents all of the regular control that a database would need. When designing your database application, you will select which ever control applies to the result you are trying to achieve. Whenever you decide to use a particular control, you should know its characteristics and limitations, the events it can send and what those events do.

Practical Learning: Starting the Application

- Start Microsoft Access and open the Fundamentals application created in the previous lesson
- 2. Open the **frmMain** form. After viewing the form, close it.

Activating a Form

A regular database application is equipped with various forms that carry appropriate data. Each form is usually linked to a particular table. Since an application can have various forms, the user sometimes needs to open many of them at the same time, to view data from different forms. Although a user can have two or more forms on screen at the same time, the user can perform any direct operation on only one form.

When two forms are present on the screen, only one of the forms can receive input from the user. The title bar of such a form displays a different color than the title bar of the other. In the following picture, compare the colors on the title bars of the Find and the Date and Time dialog boxes:

🛃 Document - WordPad		
File Edit View Insert Format I	Help	
Arial 💽 10) 💌 Western	▼ <u>B Z U </u>
<u> </u>	· 2 · · · i · · · 3 · · ·	• ! • • • 4 • • • ! • • • 5 • • • ! • • • 2
When we decided to leave, we this Find nev un: Find what: warrant	e knew we were making a h	hard decsion. We had spent so much time ? X ve wanted to make Sydney our ours. It appeared that, d Next
 Match whole word only Match case 	Date and Time Available formats: 10/6/2002 10/6/02 10/06/02 10/06/2002 02/10/06 2002-10-06 06-0ct-02 Sunday, October 06, 2002 0ctober 06, 2002 Sunday, 06 October, 2002 05 October, 2002	? × OK Cancel
For Help, press F1		

In Microsoft Access, the form that can receive input from the user is said to be active. If you want to use the other form(s), you must first make them active. To indicate the form that is active, Microsoft Access refers to the Control Panel where the Display manages colors for the applications. The color of the title bar of the active form is set by the Active Title Bar color of the Appearance property sheet.

When an application has two forms. The form that is active is, in Cartesian coordinates, above the other forms. If you click an area of another form (provided the forms are modeless or a child of the application) the other form becomes active. If you close the active form, the form that was second in Cartesian coordinates becomes active. Whenever a form becomes active, to make its presence or "activeness" be known to the operating system, the form sends an OnActive event to the operating system. If another form becomes active, the form that was active sends an OnDeActive event to the operating system, letting it know that it has lost that privilege.

Practical Learning: Making a Form Active

- 1. To create a new form, in the Forms section of the Database Window, click New
- 2. Make sure Design View is selected and click OK.
- 3. Save the form as **frmSecond** and switch it to Design View if necessary
- 4. On the Form Design toolbar, click the Code button
- 5. Verify that the title bar of the Code Editor displays Form_frmSecond. In the Object combo box, select Form.

Lesson 02: Overview of Windows Events

- 6. In the Procedure combo box, select Active
- 7. Press Tab and implement the event as follows:

```
Private Sub Form_Activate()
MsgBox "Second in Command: I am in Active duty now"
End Sub
```

- 8. Return to Microsoft Access and switch the **frmSecond** form to Form View. Notice that a message box displays
- 9. Click OK to close the message box but leave the frmSecond form open.
- 10. On the Database Window, double-click the frmMain form to open it.
- 11. In the background, click the other form to make it active. Notice that the message box displays again. Click OK

Overview of Events: The Focus Events

Microsoft Windows operating systems allow the user of your database to work on more that one form at the same time. In fact, they allow the user to work on as many forms as the computer can handle. But only one form can receive instructions at a given time. As we saw in the last exercise, only one form can be active at a time. When a form becomes active, we consider that it has received focus.

Just as an application can have many forms, a form can be equipped with various controls. Such is the case for any data entry form. On a form that is equipped with many controls, only one control can be changed at a time; such a control is said to have focus. As we saw that the form that has focus usually has its title bar with the active window color as set in Control Panel, in a form with many controls, the one that has focus will usually have a cursor or a dotted line around its selection.

When a form or a control has focus, Microsoft Visual Basic applies the **GotFocus** event to that form or control. If the focus shifts to another form or control, the control that had focus fires a **LostFocus** event and sends it to the operating system.

Practical Learning: Using an OnFocus Event

- Get back to the Code Editor. To see the structure of an OnGotFocus event, on the Procedure combo box, select GotFocus
- 2. Press Tab and implement the event as follows:

```
Private Sub Form_GotFocus()
    MsgBox "I am the focus of attention now. Give it to me"
End Sub
```

- 3. Notice that the OnGotFocus has a structure similar to the OnClick event. Get back to Microsoft Access and click the frmSecond form.
- 4. Notice that the message box for the Active event comes up first. After you click OK, the message box for the GotFocus event displays.
- 5. Click OK. Consequently, note that the Active event fires before the GotFocus event.
- 6. Get back to Microsoft Access and close the frmSecond form.
- 7. When asked whether you want to save the changes, click No.
- 8. Switch the frmMain form to Design View. If the Properties window is not displaying, double click the inside the form. If the Toolbox is not displaying, on the Form Design toolbar, click Toolbox
- 9. On the Toolbox, click Text Box and click on the form. Change the label to **First Name:** and, using the Properties window, change the name of the text box to **txtFirstName**
- 10. Add another text box to the form. Change its label to **Last Name:** and change the name of the text box to **txtLastName**
- 11. Access the Code Editor.
- 12. In the Object combo box, click txtFirstName
- 13. In the Procedure combo box, select Got Focus
- 14. Press Tab and implement it as follows:

```
Private Sub txtFirstName_GotFocus()
    txtFirstName = "I have focus"
End Sub
```

15. In the Procedure com box, select Lost Focus and implement it as follows:

```
Private Sub txtFirstName_LostFocus()
    txtFirstName = "I lost it!!!"
End Sub
```

- 16. Return to Microsoft Access. In the frmMain form, click in the First Name text box. Notice the sentence that displays in it.
- 17. Pres Tab to move the cursor away. Notice that the First Name text box displays another message

Giving Focus

As you have seen, only one form among other form can be active for the user. On a form that has many controls, only one control can receive input from the user. Two problems will regularly happen to your users while they are performing data entry: they need to have the right form or control and they need to make sure they are entering data in the intended control. At all times, it is certainly your responsibility to make these items accessible. With code, you can specify which control has focus and when. This can tremendously make the users' job easier.

To give focus to a control when another action happens, type the name of the control followed by the period operator, followed by the SetFocus procedure. An example would be:

```
Private Sub txtFirstName_GotFocus()
    txtAddress.SetFocus
End Sub
```

Keyboard Events

Word processing consists of manipulating text and characters on your computer until you get the fantastic result you long for. To display these characters, you press some keys on your keyboard. If the application is configured to receive text, your pressing actions will display characters on the screen.

The keyboard is also used to perform various other actions such as accepting what a dialog box displays or dismissing it. When the user presses the keys on a keyboard, The control in which the characters are being typed to sends one or more messages to the operating system.

There are three main events that Microsoft Windows associates to the keyboard.

• **OnKeyDown**: When the user presses a key on the keyboard, an event called KeyDown is sent to the operating system.

- OnKeyUp: When you release a key that was pressed, an even called KeyUp is sent to the operating system. These last two events apply to almost any key on your keyboard, even if the user is not typing; that is, even if the result of pressing a key did not display a character on the document.
- **OnKeyPress**: The KeyPress event is sent if the key the user pressed is recognized as a character key; that is, a key that would result in displaying a character in a document.

When programming your database, you will need to know what result you are expecting, then choose the right event. Some keys on the keyboard do not display anything on a document. Instead, they perform (only) an action. Examples of such keys are Enter, Tab, Esc. Therefore, if you mean to find out what key the user pressed, use the OnKeyDown event and not the OnKeyPress event, even though the user will have pressed a key.

Practical Learning: Keyboard Events

- 1. To see the list of keyboard events and their structures, click the arrow of the Procedure combo box and select KeyDown.
- 2. Notice the structure of the event:

Private Sub txtFirstName KeyDown(KeyCode As Integer, Shift As Integer)

End Sub

3. To call another keyboard event, on the Procedure combo box, select OnKeyPress:

Private Sub txtFirstName KeyPress(KeyAscii As Integer)

End Sub

4. In the Procedure combo box, select KeyUp and notice its event:

Private Sub txtFirstName_KeyUp(KeyCode As Integer, Shift As Integer) End Sub

Mouse Events: The MouseDown

Lesson 02: Overview of Windows Events

A mouse is equipped with buttons, usually two, that the user presses to request an action. Compared to the keyboard, the mouse claims many more events that are directly or indirectly related to pressing one of its buttons.

When the user presses one of the buttons on the mouse, an event, called OnMouseDown, is sent to the operating system. This event carries enough information for the programmer as three arguments. It is your responsibility to let the operating know what to do:

```
Private Sub txtFirstName_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

End Sub

- The operating system needs to know what button was pressed; this is represented as the left or the right button. The left button is known as vbLeftButton. The right button is referenced as vbRightButton. If the mouse is equipped with a middle button, it would vbMiddleButton. In reality, these buttons have (constant) numeric values of 0, 1, and 2 respectively.
- Secondly, the operating system needs to know whether a special key, Shift, Ctrl, or Alt, was pressed. These buttons are called vbShiftMask, vbCtrlMask, and vbAltMask respectively. In reality, they are represented with 1, 2, and 4 respectively.
- Lastly, the operating system needs to know the screen coordinates of the mouse pointer, that is, the coordinates of the point where the mouse landed. X represents the distance from the top left corner of the screen to the right direction. Y represents the vertical measure of the point from the top-left corner down.

Mouse Events: The MouseUp

When the user releases a button that was pressed on the mouse, a new event is sent to the operating system. This event is called MouseUp. It provides the same types of information as the MouseDown event:

```
Private Sub txtFirstName_MouseUp(Button As Integer, Shift As Integer,
X As Single, Y As Single)
```

End Sub

Mouse Events: The MouseMove

The MouseMove event is sent while the user is moving the mouse on the screen. It provides the same pieces of information as the MouseDown and the MouseUp events:

Lesson 02: Overview of Windows Events

```
Private Sub txtFirstName_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

End Sub

VPractical Learning: The OnMouseDown and MouseUp Events

- 1. Click the arrow of the Procedure combo box and select MouseDown
- 2. Notice the structure of the OnMouseDown event.
- 3. To initiate an OnMouseUp event, on the Procedure combo box, select MouseUp.
- 4. Close Microsoft Access

Previous

Copyright © 2002-2004 FunctionX, Inc.

<u>Next</u>



Fields, Data Types, and Variables

Introduction

Data entry on a database is performed on forms. As far as users are concerned, forms are the most important parts of a database. For a database developer, data is in fact stored in tables. Therefore, as a database developer, you should/must primarily know how tables function.

Table Design

Visually, a table is made of cells. A cell on a table is simply the intersection of a column and a row:

▦	III tblEmployees : Table							
		Employee ID	Empl #	Date Hired	Title	First Name	MI	
►	+	1	65-4386	9/26/1992	Mrs	Laurentine	D	
	+	2	35-7690	12/30/1994	Mr	Jeremiah		
	+	3	48-9349	4/14/1995	Ms	Monique	Ρ	
	+	4	34-2908	10/5/1996	Mrs	Christine	Н	
*		(AutoNumber)						
Re	Record: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1							

A column is a category of information. All data on a particular table conforms to the exact same kind of data. This means that data on a particular column can be made of text. Data on another column can be configured for numeric values. Another column can restrict the type of data that is entered on its fields.

This analysis shows us that a column in fact controls data that is entered on a table. To manage data entered under a column, Microsoft Access provides categories that control what type of data can, should, or must be entered in a cell. To specify a particular data type for a column, you must design the table that would hold the data. This is done in a special display of the table called Design View.

To display a table in Design View, if you are just designing the table, from the New Table dialog box, click Design View. If you are using Microsoft Access 2000 or 2002, you can double-click Create Table in Design View.

Microsoft Access and Visual Basic Operators

Introduction

An expression is a combination of values and symbols that allows you to get a new value; such a new value can be used in another expression, it can also be made available to a procedure, or it can be displayed in a control. For example, to get a sum of two numbers that the user is supposed to enter in a form, when creating such a form or report, you would not know what the user entered in two boxes, but you can write an expression that would use whatever the user enters, to get a new value. The values involved in an expression could be external to the program, such as algebraic numbers; they could also come from the program itself, such as a BillingRate or a DateHired fields; and they could be a combination of algebraic numbers and controls content on your program.

Comments

In the programming world, a comment is a piece of text in Visual Basic code that Visual Basic (in reality the compiler) would not consider when reading your code. As such a comment can be written any way you want.

In Visual Basic, the line that contains a comment can start with a single quote. Here is an example:

```
Private Sub Form_Load()
    ' This line will not be considered as part of the code
End Sub
```

Alternatively, you can start a comment with the **Rem** keyword. Anything on the right side of rem, Rem, or REM would not be read. Here is an example:

```
Private Sub Form_Load()
    ' This line will not be considered as part of the code
    Rem I can write anything I want on this line
End Sub
```

Comments are very useful and you are strongly suggested to use comments regularly. They can never hurt your code and they don't increase the size of your application. Comments can help you and other people who read your code to figure out what a particular section of code is used for, which can be helpful when you revisit your code after months or years of not seeing it.

The Assignment Operator =

The assignment operation is used to make a copy of a value, an expression, or the content of a control and give the copy to another field or expression. The assignment operation is performed with the = sign.

For example, suppose you have a field that displays a first name and that field is called FirstName. If you want that first name to display in another field, with this new field named, in the new field you could type:

=FirstName

On the other hand, you can use the assignment operator to give a value to a declared variable. Here is an example:

```
Private Sub Form_Load()
    Dim NumberOfTracks As Integer
    NumberOfTracks = 16
End Sub
```

When the assignment operator is provided to a variable as a starting value for the variable, this is referred to as initializing the variable.

Practical Learning: Using the Assignment Operator

- 1. Open the VBAccess1 application
- 2. Open the Assignment form
- 3. After viewing the form, switch it to Design View
- 4. On the form, click the First Name text box to select it.
- 5. On the Properties window, click the Event tab.
- 6. Double-click On Lost Focus
- 7. Right-click On Lost Focus and click Build...
- 8. Implement the event as follows:

```
Private Sub txtFirstName_LostFocus()
    ' Assign the content of the First Name text box
    ' to the Full Name text box
    txtFullName = txtFirstName
End Sub
```

- 9. Return to the form and switch it to Form View
- 10. Click the First Name text box. Type Catherine and press Tab



11. Notice that the Full Name text box got filled with the value of the First Name when the First Name text box lost focus. After using the form, switch it to Design View

The Double Quotes: ""

Double-quotes are used to display a string. First...

A string is an empty space, a character, or a group of characters that you type or provide to a control and you want this character or this group of characters to be considered "as is". In other words, the expression or the control that receives the string should keep it or them the way you supplied it or them.

A string can be an empty space or one character, such as **\$** or **w**; a group of characters, like **home** or **Manchester United** or **Verbally speaking, I mean... Ah forget it**. Most of the time, you will want the program to keep this character or group of characters exactly the way you or the user supplied them. In order to let the program know that this is a string, you must enclose it in double quotes. From our examples, our strings would be "\$", "w", "home", "Manchester United", and "Verbally speaking, I mean... Ah forget it".

To assign a string to an expression or a field, use the assignment operator as follows:

= "Manchester United"

In the same way, to initialize a variable with a string , use the assignment operator. Here is an example:

```
Private Sub Form_Load()
    Dim Address As String
    Address = "12404 Lockwood Drive Apt D4"
End Sub
```

Practical Learning: Using the Double-Quote Operator

1. On the Assign form, right-click the Full Name text box and click Build Event...



- 2. On the Choose Builder dialog box, click Code Builder and click OK
- 3. Once in the Code Editor, in the Object combo box, select cmdReset and implement its Click event as follows:

```
Private Sub cmdReset_Click()
    ' Make all text boxes empty
    txtFirstName = ""
    txtMI = ""
    txtLastName = ""
    txtFullName = ""
    txtUsername = ""
End Sub
```

- 4. Click the empty line between Private Sub txtFullName_BeforeUpdate and End Sub. Notice that the Object combo box displays txtFullName.
- 5. In the Procedure combo box, select DblClick and implement the event as follows:

```
Private Sub txtFullName_DblClick(Cancel As Integer)
   Dim strFullName As String
   strFullName = "Mary D. Lunden"
   txtFullName = strFullName
End Sub
```

6. Return to the form and double-click the Full Name text box

The String Concatenator: &

The & operator is used to append two strings, the contents of two controls, or expressions; this is considered as concatenating them. For example, it could allow you to concatenate a first name and a last name, producing a full name.

The general syntax of the concatenation operator is expressed as:

Value1 & Value2

To display a concatenated expression, use the assignment operator on the field. To assign a concatenated expression to a variable, use the assignment operator the same way:

```
Private Sub Form_Load()
   Dim FirstName, LastName As String
   Dim FullName As String
   FirstName = "Francis "
   LastName = "Pottelson"
   FullName = FirstName & LastName
   Text0 = FullName
End Sub
```

To concatenate more than two expressions, you can use as many & operators between any two expressions as necessary.

After concatenating the expressions or values, you can assign the result to another value or expression using the assignment operator. The syntax used is:

=Field1 & " " & Field2

Examples

```
=FirstName & " " & LastName
This would display, for example, Boniface Dunkirk
=[LastName] & ", " & [FirstName]
This would produce, for example, Chang, Helene
=[Address] & " " & [City] & " " & [State] & " " &
[ZIPCode] & " " & [Country]
This would display a complete address in a field
```

Practical Learning: Using the Concatenator

- 1. In the Code Editor, delete the code in the **txtFirstName_LostFocus** event
- In the Object combo box, select cmdCreateAccount and implement its Click event as follows:

Lesson 04: Microsoft Access and Visual Basic Operators

```
Private Sub cmdCreateAccount Click()
    Dim strFirstName As String
    Dim strMiddleInitial As String
    Dim strLastName As String
    Dim strFullName As String
    Dim strUsername As String
    strFirstName = txtFirstName
    strMiddleInitial = txtMI
    strLastName = txtLastName
    ' Create a username made of the last name followed by the middle initial
    strUsername = txtLastName & strMiddleInitial
    ' Create a full name as the first name followed by the last name
    strFullName = txtFirstName & " " & txtLastName
    txtFullName = strFullName
    txtUsername = strUsername
End Sub
```

- 3. Return to the form and click First Name
- 4. Type Hermine and press Tab
- 5. Type D and press Tab
- 6. In the Last Name text box, type Summers and press Tab
- 7. Notice that the first button is selected. Press Enter
- 8. Close the dialog box. When asked whether you want to save the form, click Yes.

The Negation Operator -

In mathematics, an integer such as 120 or a double floating number such as 98.005 is qualified as positive; that is, it is considered greater than 0. If a number is less than 0, to express it, you can add the - sign on the left side of the number. Examples are -5502 or -240.65. The - sign signifies that the number is negative.

A variable or an expression can also be represented as negative by prefixing it with a - sign. Examples are -Distance or -NbrOfPlayers.

To display a negative value or a negative expression in a field, use the assignment operator. Here are examples:

= -95.12= -Temperature

To initialize a variable with a negative value , use the assignment operator. Here is an example:

```
Private Sub Form_Load()
Dim NumberOfTracks As Byte
Dim Temperature As Integer
```

```
NumberOfTracks = 16
```

```
Temperature = -94
End Sub
```

The Addition: +

The addition is used to add one value or expression to another. It is performed using the + symbol and its syntax is:

Value1 + Value2

The addition allows you to add two numbers such as 12 + 548 or 5004.25 + 7.63

After performing the addition, you get a result. You can provide such a result to another variable or control. This is done using the assignment operator. The syntax used would be:

= Value1 + Value2

Practical Learning: Using the Addition

- 1. Open the frmAlgebraicOperators form.
- 2. After viewing it, switch it to Design View
- 3. The accompanying resources include pictures of geometric figures. To enhance the form, you can add them. To do that, on the Toolbox, click Image and click the left area of the labels. On the Insert Picture dialog box, locate the picture and add it.
- 4. On the form, click the Quadrilateral tab. Right-click the top Calculate button and click Build Event...
- 5. On the Choose Builder dialog box, double-click Code Builder.
- 6. In the Object combo box, select cmdRCalculate
- 7. Implement both Click events as follows:

```
Private Sub cmdRCalculate Click()
    Dim dblLength, dblHeight As Double
    Dim dblPerimeter As Double
    dblLength = txtRLength
    dblHeight = txtRHeight
    ' Calculate the perimeter of the rectangle
    ' by adding the length to the height, 2 times each
    dblPerimeter = dblLength + dblHeight + dblLength + dblHeight
    txtRPerimeter = dblPerimeter
End Sub
Private Sub cmdSqCalculate Click()
    Dim dblSide As Double
    Dim dblPerimeter As Double
    dblSide = txtSqSide
    ' Calculate the perimeter of a square by adding the side 4 times
    dblPerimeter = dblSide + dblSide + dblSide + dblSide
```

Lesson 04: Microsoft Access and Visual Basic Operators

```
txtSqPerimeter = dblPerimeter
End Sub
```

- 8. Return to the form and switch it to Form View
- 9. In the Quadrilateral tab, click Side and type 35.55
- 10. Click the top Calculate button
- 11. Click Length and type 42.72
- 12. Click Height and type 36.44
- 13. Click the other Calculate button
- 14. After using the form, return to the Code Editor

The Subtraction: -

The subtraction is performed by retrieving one value from another value. This is done using the - symbol. The syntax used is:

Value1 - Value2

The value of Value1 is subtracted from the value of Value2. After the operation is performed, a new value results. This result can be used in any way you want. For example, you can display it in a control using the assignment operator as follows:

= Value1 - Value2

The Multiplication: *

The multiplication allows adding one value to itself a certain number of times, set by the second value. The multiplication is performed with the * sign which is typed with **Shift + 8**. Here is an example:

Value1 * Value2

During the operation, Value1 is repeatedly added to itself, Value2 times. The result can be assigned to another value or displayed in a control as follows:

= Value1 * Value2

Practical Learning: Using the Multiplication
1. To apply the multiplication operation, change the Click events as follows:

```
Private Sub cmdRCalculate Click()
    Dim dblLength, dblHeight As Double
    Dim dblPerimeter As Double
    Dim dblArea As Double
    dblLength = txtRLength
    dblHeight = txtRHeight
    ' Calculate the perimeter of the rectangle
    ' by adding the length to the height, 2 times each
    dblPerimeter = dblLength + dblHeight + dblLength + dblHeight
    dblArea = dblLength * dblHeight
    txtRPerimeter = dblPerimeter
    txtRArea = dblArea
End Sub
Private Sub cmdSqCalculate Click()
    Dim dblSide As Double
    Dim dblPerimeter As Double
    Dim dblArea As Double
    dblSide = txtSqSide
    ' Calculate the perimeter of a square by adding the side 4 times
    dblPerimeter = 4 * dblSide
    dblArea = dblSide * dblSide
    txtSqPerimeter = dblPerimeter
    txtSqArea = dblArea
End Sub
```

2. Get back to the form and click both Calculate buttons:

🗉 frmAlgebraicOperators : Form 🛛 🔀					
	Algebra	uic Ope	rators		
Quadrila	Quadrilateral Circular 3-Dimensions				
		Side:	35.55	Calculate	
s	s	Perimeter:	142.20		
		Area:	1263.80		
	S				
		Length:	42.72		
н		Height:	36.40	Calculate	
		Perimeter:	158.24		
	L	Area:	1555.01		

http://www.functionx.com/vbaccess/Lesson04.htm (9 z 19) [16. 6. 2004 10:16:10]

- 3. After using the form, switch it to Design View and get back to the Code Editor
- 4. In the Object combo box, select **cmdCCalculate** and implement its Click event as follows:

```
Private Sub cmdCCalculate_Click()
   Dim dblRadius As Double
   Dim dblCircumference, dblArea As Double
   dblRadius = txtCircleRadius
   ' Circumference of a circle = 2 * Radius * PI
   dblCircumference = 2 * dblRadius * 3.14159
   ' Area of a circle = Radius * Radius * PI
   dblArea = 3.14159 * dblRadius * dblRadius
   txtCircleCircumference = dblCircumference
   txtCircleArea = dblArea
End Sub
```

- 5. Return to the form and switch it to Form View
- 6. Click the Circular tab and change the top Radius (the radius of the circle) to 64.88 and click the top Calculate button:

🖾 frmAlgebraicOperators : Form 🛛 🔀				
Algebra	ic Ope	rators		
Quadrilateral Circular 3-Din	nensions			
s s s	Side: Perimeter: Area:	64.88 ((259.52 4209.41	Calculate	
н	Length: Height: Perimeter: Area:		Calculate	

7. After using the form, switch it to Design View

The Integer Division: \

Dividing an item means cutting it in pieces or fractions of a set value. For example, when you cut an apple in the middle, you are dividing it in 2 pieces. If you cut each one of the resulting pieces, you will get 4 pieces or fractions. This is considered that you have divided the apple in 4 divisions. Therefore, the division is used to get the fraction of one number in terms of another.

Microsoft Visual Basic provides two types of results for the division operation. If you want the result of the operation to be a natural number, called an integer, use the backlash operator "\" as the divisor. Here is an example:

Value1 \ Value2

This operation can be performed on two types of valid numbers, with or without decimal parts. After the operation, the result would be a natural number.

The result of the operation can be assigned to another value. It can also be displayed in a control using the assignment operator:

= Value1 \ Value2

The Division: /

The second type of division results in a decimal number. It is performed with the forward slash "/". Its syntax is:

Value1 / Value2

After the operation is performed, the result is a decimal number. The result of either operation can be assigned to another value. It can also be displayed in a control using the assignment operator:

= Value1 / Value2

The Exponentiation: ^

Exponentiation is the ability to raise a number to the power of another number. This operation is performed using the $^{\circ}$ operator (Shift + 6). It uses the following mathematical formula:

Ух

In Microsoft Visual Basic (and Microsoft Access), this formula is written as:

y^x

and means the same thing. Either or both y and x can be values or expressions, but they must carry valid values that can be evaluated.

When the operation is performed, the value of y is raised to the power of x. You can display the result of such an operation in a field using the assignment operator as follows:

=y^x

You can also assign the operation to an expression as follows:

Total = y^x

Practical Learning: Using the Exponentiation Operator

1. Return to the Code Editor and change the code of the cmdSqCalculate_Click as follows:

```
Private Sub cmdSqCalculate_Click()
   Dim dblSide As Double
   Dim dblPerimeter As Double
   dblSide = txtSqSide
   ' Calculate the perimeter of a square by adding the side 4 times
   dblPerimeter = 4 * dblSide
   dblArea = dblSide ^ 2
   txtSqPerimeter = dblPerimeter
   txtSqArea = dblArea
End Sub
```

- 2. Get to the form and switch it to Form View.
- 3. In the Quadrilateral property sheet, enter **12.46** in the Side text box and click the top **Calculate**
- 4. After using the form, switch it to Design View

The Remainder Operator: Mod

The division operation gives a result of a number with or without decimal values, which is fine in some circumstances. Sometimes you will want to get the value remaining after a division renders a natural result. Imagine you have 26 kids at a football (soccer) stadium and they are about to start. You know that you need 11 kids for each team to start. If the game starts with the right amount of players, how many will seat and wait?

The remainder operation is performed with keyword **Mod**. Its syntax is:

Value1 Mod Value2

The result of the operation can be used as you see fit or you can display it in a control using the assignment operator as follows:

= Value1 Mod Value2

The Parentheses Operators: ()

Parentheses are used in two main circumstances: in an event (or procedures, as we will learn) or in an operation. The parentheses in an operation help to create sections in an operation. This regularly occurs when more than one operators are used in an operation.

Consider the following operation: 8 + 3 * 5

The result of this operation depends on whether you want to add 8 to 3 then multiply the result by 5 or you want to multiply 3 by 5 and then add the result to 8. Parentheses allow you to specify which operation should be performed first in a multi-operator operation. In our example, if you want to add 8 to 3 first and use the result to multiply it by 5, you would write (8 + 3) * 5. This would produce 55. On the other hand, if you want to multiply 3 by 5 first then add the result to 8, you would write 8 + (3 * 5). This would produce 23.

As you can see, results are different when parentheses are used on an operation that involves various operators. This concept is based on a theory called operator precedence. This theory manages which operation would execute before which one; but parentheses allow you to completely control the sequence of these operations.

Practical Learning: Using the Exponentiation Operator

1. Get to the Code Editor and change the following event:

```
Private Sub cmdRCalculate_Click()
   Dim dblLength, dblHeight As Double
   Dim dblPerimeter As Double
   dblLength = txtRLength
   dblHeight = txtRHeight
   ' Calculate the perimeter of the rectangle
   ' by adding the length to the height, 2 times each
   dblPerimeter = 2 * (dblLength + dblHeight)
   dblArea = dblLength * dblHeight
   txtRPerimeter = dblPerimeter
   txtRArea = dblArea
End Sub
```

2. In the Object combo box, select **cmdECalculate** and implement its event as follows:

```
Private Sub cmdECalculate_Click()
   Dim dblRadius1, dblRadius2 As Double
   Dim dblCircumference, dblArea As Double
   dblRadius1 = txtEllipseRadius1
   dblRadius2 = txtEllipseRadius2
   dblCircumference = (dblRadius1 + dblRadius2) * 3.14159
   dblArea = dblRadius1 * dblRadius2 * 3.14159
   txtEllipseCircumference = dblCircumference
   txtEllipseArea = dblArea
End Sub
```

3. Return to the form. Test the rectangle and the ellipse.

The Square Brackets Operator: []

When reviewing name rules for our objects in Microsoft Access, we saw that we should use names that are made of one word (no space). In reality, Microsoft Access, as mentioned already, is particularly flexible with names. Therefore, it uses a mechanism to delimit a name when such a name is involved in an operation.

The operator used to specify the beginning of a name is the left or opening square bracket [. At the end of a name, a right or closing bracket is used. To be safe, whether a name is made of one or more words, Microsoft prefers including it in square brackets.

Therefore, in an operation, also called an expression, instead of using such a name as FirstName, you should use [FirstName]. Of course, if a name is made of more than one word, which is frequent on databases created using the Database Wizard, you must always include it in square brackets. An example would be [Video Titles Subform]

Practical Learning: Using the Square Brackets Operator

1. Get to the Code Editor and change the following events:

```
Private Sub cmdRCalculate_Click()
   Dim dblLength, dblHeight As Double
   Dim dblPerimeter As Double
   dblLength = [txtRLength]
   dblHeight = [txtRHeight]
   ' Calculate the perimeter of the rectangle
   ' by adding the length to the height, 2 times each
   dblPerimeter = 2 * (dblLength + dblHeight)
   dblArea = dblLength * dblHeight
   [txtRPerimeter] = dblPerimeter
   [txtRArea] = dblArea
End Sub
Private Sub cmdECalculate Click()
```

```
Dim dblRadius1, dblRadius2 As Double
Dim dblCircumference, dblArea As Double
dblRadius1 = [txtEllipseRadius1]
dblRadius2 = [txtEllipseRadius2]
dblCircumference = (dblRadius1 + dblRadius2) * 3.14159
dblArea = dblRadius1 * dblRadius2 * 3.14159
[txtEllipseCircumference] = dblCircumference
[txtEllipseArea] = dblArea
End Sub
```

- 2. In the Object combo box, select cmdECalculate and implement its event
- 3. Return to the form. Test the rectangle and the ellipse

The Collection Operator: !

The objects used in Microsoft Access are grouped in categories called collections. The forms belong to a collection of objects called **Forms**. The reports belong to a collection of objects called **Reports**. Therefore, all forms of your database project belong to the **Forms** collection.

To call a particular form in an operation, using the exclamation point operator, type **Forms** followed by the **!** operator, followed by the name of the form you need to use. An example would be **Forms!ListOfEmployees** which means the ListOfEmployees form of the **Forms** collection. If the name of the form is made of more than one word, or for convenience (strongly suggested), you must use square brackets to delimit the name of the form. Therefore, the form would be access with **Forms![List Of Employees]**

If the form contains another form, which is called a subform, access it by using a second ! operator. The first ! operator as in **Forms!Employees** means the Employees form belongs to the **Forms** collection. The second ! operator would indicate that the second form (the subform) belongs to the Employees form, since the Employees form may host more than one other form. Therefore, you can access the embedded form as follows: **Forms!Employees!TimeSheet** and you can use the square brackets to delimit the name of each form as follows: **Forms![Employees]![TimeSheet]**

Practical Learning: Using the Exclamation Operator

1. Get to the Code Editor and change the code of the cmdSqCalculate_Click event as follows:

```
Private Sub cmdSqCalculate_Click()
    Dim dblSide As Double
    Dim dblPerimeter As Double
    Dim dblArea As Double
    dblSide = Forms!frmAlgebraicOperators!txtSqSide
    ' Calculate the perimeter of a square by adding the side 4 times
    dblPerimeter = 4 * dblSide
    dblArea = dblSide ^ 2
    Forms!frmAlgebraicOperators!txtSqPerimeter = dblPerimeter
    Forms!frmAlgebraicOperators!txtSqArea = dblArea
End Sub
```

2. Get back to the form and enter a value in the Side text box of the square in the Quadrilateral tab

The Line Continuation Operator:

You will regularly need to expand your code on more than two lines. This happens regularly if you are writing an expression that involves many entities that must belong to a group.

To continue a piece of code from one line to the next, type an empty space followed by an underscore symbol, then continue your code on the next line

Practical Learning: Using the Underscore Operator

- 1. In the Code Editor, click the arrow of the Object combo box and select cmdCubeCalculate_Click
- 2. Implement the Click event as follows:



3. In the Object combo box, select cmdBoxCalculate and implement its Click event as follows:

```
Private Sub cmdBoxCalculate Click()
    ' Volume = Length * Width * Height
    Forms!frmAlgebraicOperators!txtBoxVolume = ____
            Forms!frmAlgebraicOperators!txtBoxLength *
            Forms!frmAlgebraicOperators!txtBoxWidth *
            Forms!frmAlgebraicOperators!txtBoxHeight
    Dim dblLength, dblHeight, dblWidth As Double
    dblLength = Forms!frmAlgebraicOperators!txtBoxLength
    dblHeight = Forms!frmAlgebraicOperators!txtBoxWidth
    dblWidth = Forms!frmAlgebraicOperators!txtBoxHeight
    'Area = 2 * ((L * H) + (H * W) + (L * W))
    Forms!frmAlgebraicOperators!txtBoxArea = 2 * (
                                                     (dblLength * dblHeight) +
                                                     (dblHeight * dblWidth) +
                                                     (dblLength * dblWidth)
                                                 )
```

End Sub

- 4. Return to the form and switch it to Form View.
- 5. Click the 3-Dimensions tab and test box shapes:

🗉 frmAlgebraicOperators : Form 🛛 🔀				
Algebrai	Algebraic Operators			
Quadrilateral Circular 3-Dime	nsions			
s	Side: Area: Volume:	25.55 3916.82 16679.10	Calculate	
H	Length: Width: Height: Area: Volume:	14.55 10.75 12.85 963.03 2009.90	Calculate	

6. Save the form and close it

The Period Operator:.

In the next lesson, we will learn about properties of an object. We will find out that a property is something that describes an object. For example, users mainly use a text box either to read the text it contains, or to change its content, by changing the existing text or by entering new text. Therefore, the text the user types in a text box is a property of the text box.

To access the property of an object, type the name of the object, followed by a period, followed by the name of the property you need. The property you are trying to use must be part of the properties of the object. In Microsoft Access, to use a property of an object, you must know, either based on experience or with certainty, that the property exists. In Visual Basic, the Code Editor is very helpful and would display the list of properties once you type the period:

-43	🕸 Form_Assignment : Class Module 🛛 📃 🗖 🔀		
F	orm	▼ Load	•
	Option C Option E	ompare Database xplicit	▲
	Private : Deta End Sub	Sub Form_Load() il.	
		 BackColor BackColor CanGrow CanShrink 	
		reenter Controls reenter EventProcPrefix reenter refixer refi	

If you know the name of the property, you can start typing it. Once the desired property is highlighted, press the Space bar or Tab.

If you see the name of the property in the list, you can double-click click it.

If the list doesn't appear, which will happen sometimes for any reason (with experience, you will find out why), press Ctrl + Space bar.

If you don't want to use the list displayed by the Code Editor, press Esc.

Once you have specified what property you want to use, you can assign it the desired value or you can involve it in any operation you see fit.

Constants

A constant is a value that doesn't change (this definition is redundant because the word value already suggests something that doesn't change). There are two types of constant you will use in your programs: those supplied to you and those you define yourself.

The Carriage Return-Line Feed Constant

Visual Basic provides the vbCrLf constant. It is used to interrupt a line of text and move to the next line.

Built-in Constants: PI

PI is a mathematical constant whose value is approximately equal to 3.1415926535897932. It is highly used in operations that involve circles or geometric variants of a circle: cylinder, sphere, cone, etc.

Built-in Logical Constants: NULL

A variable is said to be null when its value is invalid or doesn't bear any significant or recognizable value.

Built-in Logical Constants: TRUE and FALSE

An expression is said to be false if the result of its comparison is 0. Otherwise, the expression is said to bear a true result.

Previous

Copyright © 2002-2004 FunctionX, Inc.

Next

III Table1 : Table		
Field Name	Data Type	Description
•		
		-1
	Field Properties	
General Lookup		
		A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Once in Design View, you must specify a name for each field

Fields Names

Each field on a database must have a name (this is also true for anything in the computer). The name held by a field allows you, the database developer, and the operating system, to refer to that particular field.

To specify the name of a field, type a name in the Field Name column of the table in Design View (The name of a field can also be specified in <u>Datasheet View</u>). There are rules and suggestions used to specify a name for a field. A field name

- Must begin with a letter (a-z or A-Z)
- Cannot contain an period (.) or a special character
- Must not contain a space
- Must not exceed 255 characters. You should limit the name of a variable to 30 characters
- Must be unique in the same scope

A field can have almost any name such as "first name", "Employee's first name", "First name of the employee", etc.

On our database, that is, on the fields we create here:

- The name of each field will be made of one word, no space
- The name of a field will start in uppercase; examples are Address, Category, or Format
- If the name of a field is a combination of words, such as First and Name, or Date and Of and Birth, instead of writing the whole name indistinctly, each component of the common name will start in uppercase. Instead of Firstname, we will use FirstName. Instead of Dateofbirth, we will use DateOfBirth.

After specifying a name of the field, you can decide what type of data can be entered into that field. The types of data are organized in categories. To specify a data type, click under the Data Type column. This displays an arrow that indicates that this column is in fact made of combo boxes:

Tables Data Types

A **Text** data type allows the user to type any kind of character or group of characters. The field can hold up to 255 characters.

The **Memo** data type functions like the Text except that it can hold up to 64000 characters.

The **Number** data type allows a combination of digits using 0 to 9. As you will see shortly, there are various types of numbers. You will get to specify further number formats you need for your database.

The **Date/Time** data type allows you to control data entry to some fields for recognizable date or time values. There are various formats used.

The **Currency** data type will accept only a number in its field. That number is then converted to represent money value.

You use the **AutoNumber** data type to ask Microsoft Access to complete that field with automatically generated numbers for a field whose numbers you don't need to control.

An **OLE Object** field allows you to include pictures, graphics, other application files (for example Microsoft Word documents or Microsoft Excel spreadsheets, etc).

With the **Hyperlink**, you can provide a link to a file in your computer or to a web page on the Internet.

The **Lookup Wizard** specifies some predefined data on a particular column. These data are usually prepared by you.

Tabular Description of Data Types

Data Type	Used For	Examples
Toxt	Group of characters of any kind, but it is	AC Milan
ΤΕΧΙ	limited to 255 characters	John
Memo	Larger text with a maximum of 64000 characters	A text document
Number	Numeric data	212
Data/Tima	Data ar tima	12/26/90
Date/ Time		1:00 PM
Currency	Currency (money) value	\$ 34,000
AutoNumber	Setting numeric values automatically	1, 2, 3, 4
OLE Object	Other kinds of files in your database	
Hyporlink	linking from your document to a file in your	www.pbs.org
пуреннк	computer or to a web page on the Internet	www.atl.com
Lookup Wizard	Predefined data on a particular column. These data are usually prepared by you	

Practical Learning: Designing a Table

- 1. Start Microsoft Access and open the **CPAP Help Desk1** database that you created previously.
- 2. If the Database Window is not present, press F11. In the Database Window, click Tables. In the Tables section or Tables property sheet, click the New button
- 3. In the New Table dialog dialog box, click Design View and click OK.
- 4. Notice that the first field under Field Name has focus. Type **TimeSheetID** and press Tab
- 5. Click the arrow of the Data Type combo box and select **AutoNumber**

🔠 Table1 : Table		
Field Name	Data Type	Description
TimeSheetID	Text	•
	Text	
	Memo	
	Number	
	Date/Time	
	Currency	
	AutoNumber	
	Yes/No *V	
	Lookup Wizard	· · · · · · · · · · · · · · · · · · ·
	Field	TProperties
General Lookup Field Size Format Input Mask Caption Default Value Validation Rule Validation Text Required Allow Zero Length Indexed Unicode Compression	50 No No Yes (Duplicates OK) Yes	The data type determines the kind of values that users can store in the field. Press F1 for help on data types.

- 6. Right-click **TimeSheetID** and click Primary Key
- 7. Click the first field under **Description** and type **Automatic number set by the** database
- 8. To save the table, on the Table Design toolbar, click the Save button \blacksquare .

- 9. Type tblTimeSheet and press Enter.
- 10. Click the first field under TimeSheetID and type EnteredBy
- 11. Press the down arrow key. Notice that the Data Type for the **EnteredBy** field is set to Text.
- 12. Type **DateEntered** and press Enter
- 13. In the Data Type combo box, select **Date/Time**
- 14. Complete the table as follows:

▦	tblTimeSheet : Table		<u>_ </u>]	×
	Field Name	Data Type	Description	
8	TimeSheetID	AutoNumber	Automatic number set by the database	
	EnteredBy	Text	The employee who filled the time sheet	
	DateEntered	Date/Time	The date this time sheet was filled	
	NumberOfHours	Number	The total number of hours worked for the week	
	Salary	Text	The hourly salary	
	MaritalStatus	Text	If the employee is married, click the check box	
	Notes	Text	A few notes or comments about this time sheet	
				-
_		Field P	roperties	
	General Lookup		A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.	

- 15. To switch the table to the other view, on the main menu, click View -> Datasheet View.
- 16. When asked whether you want to save the table, click Yes.
- 17. After viewing the table, close it.

Introduction to Variables

If you create a simple desktop database, the data types and fields in Microsoft Access are sufficient. To customize the behaviors of your fields for a functionality you cannot implement in Microsoft Access, you will use Microsoft Visual Basic as a programming language.

The computer memory is made of small storage areas used to hold the things or components that a program needs while it is running. As a programmer, you specify these things, or you provide them to the computer; the computer then puts them in its storage areas. When you need one of them, you let the computer know. The machine "picks it up", brings it to you, and then you can use it as you see fit.

In the world of computer programming, a variable is a value you ask the computer to temporarily store in its memory while the program is running.

Declaring a Variable

In order to reserve that storage area, you have to let the computer know. Letting the computer know is referred to as *Declaring* the variable. You declare a variable using the **Dim** keyword, like this:

Dim

Just like any of the controls we have used so far, a variable needs to have a name. There are rules you should follow when naming your variables. The name of a variable:

- · Must begin with a letter
- After starting with a letter, can be made of letters, underscores, and digits in another order
- Cannot have a period
- Can have up to 255 characters.
- Must be unique inside of the event (or procedure, function or module (we will learn what these things are)) it is used in.

To name a variable, follow the same rules and suggestions we used for a table's fields.

To declare a variable, locate the event that you want to use the variable in. Follow the above syntax and the below suggestions and data types.

Visual Basic Data Types

Just like you specify a data type for a table's field, you also specify a data type for a variable. This time, a data type tells the computer what kind of variable you are going to use in your program. There are different kinds of variables for various purposes. Before assigning a data type to a variable, you should know how much space a data type will occupy in memory. Different variables or different data types use different amount of space in memory. The amount of space used by a data type is measured in bytes.

Most of the data types you use in Visual Basic correspond to those you used in Microsoft Access and are easily recognizable.

String

A **String** in Visual Basic is used for characters such as names and text of any kind and almost any length. It can be used for a Microsoft Access **Text** or **Memo** data types. To declare a string data type, for example when a form opens, here is an example:

```
Private Sub Form_Load()
Dim CountryName As String
End Sub
```

Boolean

In Microsoft Access, you can specify a field as providing a Yes or No, a True or False, an On or Off, a 1 or 0 values. Its Microsoft Access corresponding type is the **Boolean** data type. Here is an example of declaring a Boolean variable when the form opens:

```
Private Sub Form_Load()
Dim CountryName As String
Dim IsMarried As Boolean
End Sub
```

Byte

If you are planning to use a number in your program, you have a choice from different kinds of numbers that Visual Basic can recognize. You can use the same **Byte** data type which is a natural number that ranges from 0 to 255. You can declare it as follows:

```
Private Sub Form_Load()
Dim CountryName As String
Dim IsMarried As Boolean
Dim StudentAge As Byte
```

Integer

An **Integer** is a number that ranges from -32,768 to 32,767. It is the same Integer used in Microsoft Access. It is the same integer used in Microsoft Access. The integer type should always be used when counting things such as books in a library or students in a school; in this case you would not use decimal values. Here is an example of declaring an integer variable:

```
Private Sub Form_Load()
Dim CountryName As String
Dim IsMarried As Boolean
Dim StudentAge As Byte
Dim Tracks As Integer
End Sub
```

Long

A **Long** integer is a number that can be used for a field or variable involving greater numbers than integers. The **Long** type is also suitable for numbers that don't involve decimal values. You can use the **Long** type as corresponding to the **AutoNumber** used in Microsoft Access (as long as you are not using it as Replication ID), although the **AutoNumber** was a **Long Integer**. Here is an example where a **Long** integer variable is declared:

```
Private Sub Form_Load()
   Dim CountryName As String
   Dim IsMarried As Boolean
   Dim StudentAge As Byte
   Dim Tracks As Integer
   Dim Population As Long
End Sub
```

Single

A **Single** is a decimal floating number. Unlike an integer or a **Long** integer, a **Single** variable can have a decimal portion such as 12.44 or 804.60. Here is an example of declaring a **Single** floating variable:

```
Private Sub Form_Load()
Dim CountryName As String
Dim IsMarried As Boolean
Dim StudentAge As Byte
Dim Tracks As Integer
```

Lesson 03: Fields, Data Types, and Variables

```
Dim Population As Long
Dim Distance As Single
End Sub
```

Double

A **Double** data type should be used for a variable that carries (or would carry) a decimal value. This data type is similar to Microsoft Access' Double.

```
In most circumstances, it is preferable to use Double
instead of Single when declaring a variable that would
hold a decimal number. Although the Double takes more
memory spaces (computer memory is not expensive
anymore(!)), it provides more precision.
```

Here is an example of declaring a Double variable:

```
Private Sub Form_Load()

Dim Distance As Double

End Sub
```

Currency

The **Currency** data type is used to deal with currency values the same way the Currency is used in MS Access. Here is an example of declaring it:

```
Private Sub Form_Load()
   Dim CountryName As String
   Dim IsMarried As Boolean
   Dim StudentAge As Byte
   Dim Tracks As Integer
   Dim Population As Long
   Dim Distance As Single
   Dim StartingSalary As Currency
End Sub
```

Date

In Microsoft Access, a Date/Time data type is used to control a field with a date or a time values. In Visual Basic, a **Date** data type is used to specify a date or time value. Therefore, in Microsoft Visual Basic, to declare either a date or a time variables, use the **Date** keyword. Here are two examples:

```
Private Sub Form_Load()
Dim CountryName As String
Dim IsMarried As Boolean
```

Lesson 03: Fields, Data Types, and Variables

```
Dim StudentAge As Byte
Dim Tracks As Integer
Dim Population As Long
Dim Distance As Single
Dim StartingSalary As Currency
Dim DateOfBirth As Date
Dim KickOffTime As Date
End Sub
```

Object

An **Object** is almost anything else that you want to use in your program. If you don't specify a data type or can't figure out what data type you want to use, you can use the Variant or let Visual Basic use the Variant data type.

Variant

A **Variant** can be used to declare any kind of variable. You can use a variant when you can't make up your mind regarding a variable but, as a beginning programmer, you should avoid it.

Here is a table of various data types and the amount of memory space each one uses:

Data type	Description	Range
Byte	1-byte binary data	0 to 255
Integer	2-byte integer	- 32,768 to 32,767
Long	4-byte integer	- 2,147,483,648 to 2,147,483,647
Single	4-byte floating-point	 - 3.402823E38 to - 1.401298E - 45 (negative values) 1.401298E - 45 to 3.402823E38 (positive)
		values)
Double	8-byte floating-point	– 1.79769313486231E308 to – 4.94065645841247E – 324 (negative values)
	number	4.94065645841247E – 324 to 1.79769313486231E308 (positive values)
Currency	8-byte number with fixed decimal point	- 922,337,203,685,477.5808 to 922,337,203,685,477.5807
String	String of characters	Zero to approximately two billion characters

Variant	Date/time, floating-point number, integer, string, or object. 16 bytes, plus 1 byte for each character if a string value.	Date values: January 1, 100 to December 31, 9999 Numeric values: same range as Double String values: same range as String Can also contain Error or Null values
Boolean	2 bytes	True or False
Date	8-byte date/time value	January 1, 100 to December 31, 9999
Object	4 bytes	Any Object reference

We have learned how to declare a variable as follows:

Dim CountryName As String

We also saw that we can declare different variables each on its own line as follows;

Dim FirstName **As** String **Dim** LastName **As** String **Dim** Salary **As** Currency **Dim** AlreadyVisited **As** Boolean

If you have many variables of the same data type, you can declare them on the same line, each separated with a comma; remember to specify their common type. Here is an example:

Private Sub Form_Load() **Dim CountryName, Address, City, State As String** Dim IsMarried As Boolean Dim StudentAge As Byte Dim Tracks As Integer Dim Population As Long Dim Distance As Single **Dim StartingSalary, WeeklyEarnings As Currency Dim DateOfBirth, KickOffTime As Date** End Sub

When naming your variables, besides the above suggestions, you can start a variable's name with a one to three letter prefix that could identify the data type used. Here are a few suggestions:

Lesson 03: Fields, Data Types, and Variables

Data Type	Prefix	Example
Boolean	bln	blnFound
Byte	byt	bytTracks
Date/Time	dtm	dteStartOfShift
Double	dbl	dblDistance
Error	err	errCantOpen
Integer	int	intNbrOfStudents
Long	Ing	IngPopulation
Object	obj	objConnection
Single	sng	sngAge
String	str	strCountryName
Currency	cur	curHourlySalary
Variant	var	varFullName

Previous

Copyright © 2002-2004 FunctionX, Inc.

<u>Next</u>



Modules, Procedures, and Functions

Modules

A module is file that holds code or pieces of code that belong either to a form, a report, or is simply considered as an independent unit of code. This independence means that a unit may also not belong to a particular form or report. Each form or report has a (separate) module. To access the module of a form or report, open the object in Design View and click the Code

button 2. If you initiate the coding of an event of a form (or report) or of a control that is positioned on the form, this would open the Code Editor window and display the module that belongs to the form (or report). If no code has been written for the form or report, its module would be empty:



If you have written events for a form (or report) or for the controls that belong to the form (or report), all these events would be part of the form's (or report's) module.

Besides the modules that belong to forms (or reports), you can create your own module that are not related to a particular form (or report). There are three main ways you can create an independent module:

- In Microsoft Access, on the Database Window, click Modules and click New. This would open the Code Editor with an empty file.
- In the Code Editor (or Microsoft Visual Basic) window, on the main menu, click Insert -> Module
- In the Code Editor (or Microsoft Visual Basic) toolbar, you can click the Insert Module button or click the arrow of the Insert Module button and select Module.

The name of modules are cumulative. This means that the first module would be called called Module1; the second would be Module2, etc. It is a good idea to have names that are explicit especially if your application ends up with various modules. To use a custom name for a

module, you must save it. This would prompt you to name the module. You can accept the suggested name or type your own and press Enter.

Practical Learning: Creating a Module

- 1. Start Microsoft Access and open the Bethesda Car Rental3 database
- 2. On the Database Window, click Modules and click New
- 3. To save the module, on the toolbar, click the Save button \blacksquare
- 4. Type **modAssignments** and press Enter. Notice the name of the module on the title bar.
- 5. Open the VBAccess1 application. On the Database Window, if necessary, click Forms
- 6. Double-click the **frmProcedures** form to open it in Form View. After viewing the form, switch it to Design View
- 7. The accompanying resources include pictures of geometric figures. To enhance the form, you can them. To do that, on the Toolbox, click Image and click the left area of the labels. On the Insert Picture dialog box, locate the picture and add it.
- 8. To start a form module, with the form opened in Design View, on the Form Design toolbar, click the Code button

Sub Procedures

A procedure is an assignment you ask Microsoft Visual Basic to perform besides, or to complete, the normal flow of the program. A procedure is created to work in conjunction with the controls events of a database. Structurally, a procedure appears similar to an event. The main difference is that, while an event belongs to a control, a procedure doesn't. While an event is specific to the user's intervention or interaction with a control, a procedure can be general and applied anyway you like.

Creating a Procedure Manually

There are two kinds of procedures in Microsoft Visual Basic: A sub procedure and a function. The difference lies on their behaviors but their coding (programming) depends of your goal.

A sub procedure is a section of code that carries an assignment but doesn't give back a result. To create a sub procedure, start the section of code with the **Sub** keyword followed by a name for the sub procedure. To differentiate the name of the sub procedure with any other regular name, it must be followed by an opening and closing parentheses. The **Sub** keyword and the name of the procedure (including its parentheses) are written on one line (by default). The section of the sub procedure code closes with **End Sub** as follows:

Sub ShowMe()

End Sub

The name of a sub procedure should follow the same rules we learned to name the variables, omitting the prefix:

- If the sub procedure performs an action that can be represented with a verb, you can use that verb to name it. Here are examples: show, display
- To make the name of a sub procedure stand, you should start it in uppercase. Examples are Show, Play, Dispose, Close
- You should use explicit names that identify the purpose of the sub procedure. If a
 procedure would be used as a result of another procedure or a control's event, reflect it on
 the name of the sub procedure. Examples would be: afterupdate, longbefore.
- If the name of a procedure is a combination of words, start each word in uppercase. Examples are: AfterUpdate, SayItLoud

The section between the first Sub line and the End Sub line is called the body of the sub routine. In the body of a procedure, you define what the procedure is supposed to do. If you need to use a variable, you can declare it and specify what kind of variable you need. There is no restriction on the type of variables that can be declared in a procedure. Here is an example in which a string variable is declared in the body of a sub procedure:

```
Sub CreateName()
Dim strFullName As String
End Sub
```

In the same way, you can declare as many variable as you need inside of a sub procedure. The actions you perform inside of a procedure depend on what you are trying to accomplish. For example, a sub procedure can be simply be used to create a string. The above procedure can be changed as follows:

```
Sub CreateName()
    Dim strFullName As String
    strFullName = "Jacques Fame Ndongo"
End Sub
```

Similarly, a procedure can be used to perform a simple calculation such as adding two numbers. Here is an example:

```
Sub CalculateTotalStudents()
   Dim StudentsInClass1 As Integer
   Dim StudentsInClass2 As Integer
   Dim TotalNumberOfStudents As Integer
   StudentsInClass1 = 32
   StudentsInClass2 = 36
   TotalNumberOfStudents = StudentsInClass1 + StudentsInClass2
End Sub
```

There are two main ways a sub procedure receives values. To start, a procedure that is written in the module of a form (or report) has direct access to the controls that belong to the form (or report). This means that the procedure can call them and manipulate any of their available properties. Here is an example of a procedure implemented in a form that has a text box called txtCountry: Lesson 5: Modules, Procedures, and Functions

```
Sub ChangeColor()
    txtCountry.BackColor = 16763293
End Sub
```

In the same way, you can declare variables and perform operations inside of a procedure and hand the result to a control that is part of a form or report.

Practical Learning: Manually Creating a Procedure

- 1. Click the first empty line in the Code Editor and type Sub SquareSolution and press Enter
- 2. Notice that Visual Basic added the End Sub line and positioned the cursor inside the procedure.
- 3. Complete the sub procedure as follows:

```
Sub SquareSolution()
    ' Declare the necessary variables for the square
    Dim dblSide As Double
    Dim dblPerimeter, dblArea As Double
    ' Retrieve the value of the side
    dblSide = txtSqSide
    ' Calculate the perimeter and the are of the square
    dblPerimeter = dblSide * 4
    dblArea = dblSide * dblSide
    ' Prepare to display the result in the appropriate text boxes
    txtSqPerimeter = dblPerimeter
    txtSqArea = dblArea
End Sub
```

4. Return to Microsoft Access and switch the form to Form View. Enter a value in the Side text box of the square and click the Calculate button. Notice that nothing happens.

Inserting Procedure

Microsoft Visual Basic simplifies the creation of a procedure through the use of the Insert Procedure dialog box:

Insert Proced	ure	<u>? ×</u>
Name:		ОК
C Sub	• Function C Property	Cancel
Scope • Public	C Private	
All Local Va	ariables as Statics	

To display the Insert Procedure, you can click Insert -> Procedure on the main menu, or click the arrow of the Insert Procedure button on the toolbar. If you are creating a sub procedure,

click the Sub radio button. If you want the procedure to be access only by the objects, events and procedure of the same module, click the Private radio button. If you want to access the procedure from outside of the current module, click the Public radio button.

Practical Learning: Inserting a Procedure

- 1. Switch the form back to Design View and access the Code Editor. On the main menu, click Insert -> Procedure...
- 2. On the Insert Procedure dialog box, click the Name text box and type SolveRectangle
- 3. In the Type section, click the Sub radio button.
- 4. In the Scope section, click the Private radio button

Insert Procedure	? 🔀
Name: SolveRectangle	ОК
Type Sub C Function C Property	Cancel
Scope O Public O Private	
All Local Variables as Statics	

- 5. Click OK
- 6. Implement the procedure as follows:

```
Private Sub SolveRectangle()
    ' Declare the necessary variables for the rectangle
    Dim dblLength, dblHeight As Double
    Dim dblPerimeter, dblArea As Double
    ' Retrieve the values of the length and height
    dblLength = txtRLength
    dblHeight = txtRHeight
    ' Calculate the perimeter and the area of the rectangle
    dblPerimeter = (dblLength + dblHeight) * 2
    dblArea = dblLength * dblHeight
    ' Prepare to display the result in the appropriate text boxes
    txtRPerimeter = dblPerimeter
    txtRArea = dblArea
End Sub
```

Calling a Procedure

After creating a procedure, you can call it from another procedure, function, or control's event. To call a simple procedure such as the above ChangeColor, you can just write the name of the sub procedure. Here is an example where a sub procedure is called when a form is clicked. Lesson 5: Modules, Procedures, and Functions

```
Private Sub Detail_Click()
    ChangeColor
End Sub
```

Practical Learning: Calling a Procedure

- 1. In the Object combo box, select cmdSqCalculate
- 2. Call the SolveSquare procedure as follows:

```
Private Sub cmdSqCalculate_Click()
    SquareSolution
End Sub
```

3. In the Object combo box, select cmdRCalculate and implement its Click event follows:

```
Private Sub cmdRCalculate_Click()
    SolveRectangle
End Sub
```

- 4. Return to Microsoft Access and switch the form to Form View.
- 5. In the Side text box of the square, type 14.55 and click the right Calculate button
- 6. Also text the rectangle with 18.25 for the length and 15.75 for the height and click its Calculate button:

🖽 frmPro	ocedures : Forn	n		
	Algebra	aic Ope	erators	
Quadrilat	eral Circular 3-D	imensions		
S	s	Side: Perimeter: Area:	14.55 (58.20 211.70	Calculate
н	L	Length: Height: Perimeter: Area:	18.25 15.75 68.00 287.44	Calculate

7. Save the form and return to the Code Editor

Function Procedures

A function is a procedure that takes care of a relatively small assignment on a side and returns a result. A function resembles a sub procedure in all respects except that a function returns a value.

Creating a Function

A function is created like a sub procedure with a few more rules. The creation of a function starts with the keyword **Function** and closes with **End Function**. Here is an example:

```
Function GetFullName()
```

```
End Function
```

The name of the function follows the same rules and suggestions we have reviewed for the sub procedures. Because a function should return a value, after the parentheses, type the **As** keyword followed by the type of data the function must return. Here is an example:

```
Function GetFullName() As String
```

End Function

The implementation of a function is done the same way that of a sub procedure is. Because a function is supposed to return a value, after performing whatever assignment you need in a function, you can assign the desired result to the name of the function before the closing of the function. Here is an example:

	First Name: Last Name: Full Name:	Jacob Moonstruck Jacob Moonstruck
Reco	ord: 🚺 🔳	1 ▶ ▶ ▶ ▶ ★ of

```
Dim strFirstName() As String
Dim strFirstName, strLastName As String
strFirstName = txtFirstName
strLastName = txtLastName
GetFullName = strFirstName & " " & strLastName
End Function
```

Practical Learning: Creating a Function

- 1. In the Code Editor, scroll down, click the first empty line, type **Function CircleCircumference As Double** and press Enter
- 2. Notice that Visual Basic completed the code with the **End Function** line and positioned the cursor in the body of the function. Implement the function as follows:

Lesson 5: Modules, Procedures, and Functions

```
Function CircleCircumference() As Double
Dim dblRadius As Double
dblRadius = txtCircleRadius
CircleCircumference = dblRadius * 2 * 3.14159
End Function
```

- 3. On the main menu, click Insert -> Procedure...
- 4. Type CircleArea in the Name text box
- 5. In the Type section, make sure the **Function** radio is selected (the default). In the Scope section, click the **Private** radio button and click OK.
- 6. Implement the function as follows:

```
Private Function CircleArea() As Double
    Dim dblRadius As Double
    dblRadius = txtCircleRadius
    CircleArea = dblRadius * dblRadius * 3.14159
End Function
```

Calling a Function

To call a function, you have two main alternatives. If the function was implemented as simple as a sub procedure, you can just write its name in the event or the function that is calling it. If you want to use the return value of a function in an event or another function, assign the name of the function to the appropriate local variable. Here is an example:

```
Private Sub Detail_DblClick(Cancel As Integer)
    txtFullName = GetFullName
End Sub
```

Practical Learning: Calling a Function

1. In the Object combo box, select cmdCCalculate and implement its Click event as follows:

```
Private Sub cmdCCalculate_Click()
    txtCircleCircumference = CircleCircumference
    txtCircleArea = CircleArea
End Sub
```

- 2. Switch to Microsoft Access and switch the form to Form View.
- Click the Circular tab and, in the top Radius text box, type 25.55
- 4. Click the right calculate button (for the Circle).

Accessories for Programming

Introduction

When using a database, you are in fact using two applications to create a final product. Microsoft Access is used to design the necessary objects for your product. This means that Microsoft Access is used for its visual display of objects. On the other hand, Microsoft Visual Basic is used to handle code that enhance the functionality of your application.

The Compiler

The code you write is made of small instructions written in Visual Basic. These instructions are written in English, a language that the computer, that is the operating system, doesn't understand. Visual Basic, as its own language among other computer languages, is internally equipped with a low level program called a compiler. This program takes your English language instructions and translates them in a language the computer can understand. The language the computer speaks is known as the machine language. You usually don't need to know anything about this language.

After writing your code, at one time it is transmitted to the compiler. The compiler analyzes it first, checks its syntax, the words used in the program, the variables are checked for their declaration and use. The events and procedures are checked for their behavior. The expressions are checked for their accuracy. If something is wrong with the code, that is, if the compiler does not understand something in your code, it would display an error and stop. You must correct the mistake or else... As long as the compiler cannot figure out a piece of code in a module, it would not validate it. If the code is "admissible", the compiler would perform the assignments that are part of the code and give you a result based on its interpretation of the code. This means that the code can be accurate but produce an unreliable or false result. This is because the compiler is just another program: it does not think and does not correct mistakes although it can sometimes point them out. For this reason, you should know what you are doing.

Writing Argumentative Procedures

To carry out an assignment, sometimes a procedure needs one or more values to work on. If a procedure needs a value, such a value is called an argument. While a certain procedure might need one argument, another procedure might need many arguments. The number and types of arguments of a procedure depend on your goal.

An argument is also called a parameter. Both words mean the same thing.

If you are writing your own procedure, then you will decide how many arguments your procedure would need. You also decide on the type of the argument(s). For a procedure that is taking one argument, inside of the parentheses of the procedure, write the name of the argument followed by the **As** keyword followed by the type of data of the argument. Here is an example:

Sub CalculateArea(Radius As Double)

End Sub

A procedure can take more than one argument. If you are creating such a procedure, between the parentheses of the procedure, write the name of the first argument followed by **As** followed by the data type, followed by a comma. Add the second argument and subsequent arguments and close the parentheses. There is no implied relationship between the arguments; for example, they can be of the same type:

Sub CalculatePerimeter (Length As Double, Height As Double)

End Sub

The arguments of your procedure can also be as varied as you need them to be. Here is an example:

Sub DisplayGreetings (**strFullName As String**, **intAge As Integer**, **dblDistance As Double**)

End Sub

Practical Learning: Writing Procedures With Arguments

1. Switch to the Code Editor. Click an empty area at the end of the existing code and create the following procedure:

```
Sub SolveEllipse(SmallRadius As Double, LargeRadius As Double)
Dim dblCircum As Double
dblCircum = (SmallRadius + LargeRadius) * 2
dblArea = SmallRadius * LargeRadius * 3.14159
txtEllipseCircumference = dblCircum
txtEllipseArea = dblArea
End Sub
```

To create an example of function that takes an argument, add the following function at the end of the existing code:

```
Function CubeArea(Side As Double) As Double
CubeArea = Side * Side * 6
End Function
```

To use different examples of functions that take one or two arguments, type the following functions:

```
Function CubeVolume(Side As Double) As Double
CubeVolume = Side * Side * Side
End Function
```

Lesson 5: Modules, Procedures, and Functions

```
Function BoxArea(dblLength As Double,
                 dblHeight As Double,
                 dblWidth As Double) As Double
    Dim Area As Double
    Area = 2 * ((dblLength * dblHeight) + _
                (dblHeight * dblWidth) + _
                (dblLength * dblWidth) _
               )
    BoxArea = Area
End Function
Function BoxVolume(dblLength As Double,
                dblHeight As Double,
                 dblWidth As Double) As Double
    Dim Volume As Double
    Volume = dblLength * dblHeight * dblHeight
    BoxVolume = Volume
```

End Function

Calling Argumentative Procedures

We saw already how to call a procedure that does not take any argument. Actually, there are various ways you can call a sub procedure. As we saw already, if a sub procedure does not take an argument, to call it, you can just write its name. If a sub procedure is taking an argument, to call it, type the name of the sub procedure followed by the name of the argument. If the sub procedure is taking more than one argument, to call it, type the name of the procedure followed by the name of the arguments, in the exact order they are passed to the sub procedure, separated by a comma. Here is an example:

```
Private Sub txtResult GotFocus()
    Dim dblHours As Double
    Dim dblSalary As Double
    dblHours = txtHours
    dblSalary = txtSalary
    CalcAndShowSalary dblHours, dblSalary
End Sub
Sub CalcAndShowSalary(Hours As Double, Salary As Double)
    Dim dblResult As Double
    dblResult = Hours * Salary
    txtResult = dblResult
End Sub
```

Alternatively, you can use the keyword **Call** to call a sub procedure. In this case, when calling a procedure using **Call**, you must include the argument(s) between the parentheses. using Call, the above GotFocus event could call the CalcAndShowSalary as follows:

```
Private Sub txtResult GotFocus()
    Dim dblHours As Double
```

Lesson 5: Modules, Procedures, and Functions

```
Dim dblSalary As Double
dblHours = txtHours
dblSalary = txtSalary
Call CalcAndShowSalary(dblHours, dblSalary)
End Sub
```

Practical Learning: Calling Procedures With Arguments

 To call the above procedures that take arguments, on the Object combo box, select cmdECalculate and implement its OnClick event as follows:

```
Private Sub cmdECalculate_Click()
   Dim Radius1 As Double
   Dim Radius2 As Double
   Radius1 = txtEllipseRadius1
   Radius2 = txtEllipseRadius2
   SolveEllipse Radius1, Radius2
End Sub
```

On the Object combo box, select cmdCubeCalculate and implement its Click event as follows:

```
Private Sub cmdCubeCalculate_Click()
   Dim dblSide As Double
   Dim dblArea As Double
   Dim dblVolume As Double
   dblSide = txtCubeSide
   dblArea = CubeArea(dblSide)
   dblVolume = CubeVolume(dblSide)
   txtCubeArea = dblArea
   txtCubeVolume = dblVolume
End Sub
```

On the Object combo box, select cmdBoxCalculate and implement its Click event as follows:

```
Private Sub cmdBoxCalculate_Click()
   Dim dLen As Double
   Dim dHgt As Double
   Dim dWdt As Double
   Dim Area, Vol As Double
   dLen = txtBoxLength
   dHgt = txtBoxHeight
   dWdt = txtBoxWidth
   Area = BoxArea(dLen, dHgt, dWdt)
   Vol = BoxVolume(dLen, dHgt, dWdt)
```

```
txtBoxArea = Area
txtBoxVolume = Vol
End Sub
```

- 4. Close the Code Editor or Microsoft Visual Basic and return to Microsoft Access.
- 5. Switch the form to Form View and test the ellipse in the Circular tab.
- 6. Also test the cube and the box in the 3-Dimensions tab

🕫 frmProcedures : Form		×
Algebra	ic Op	erators
Quadrilateral Circular 3-Dim	ensions	
s	Side: Area:	28.12 Calculate
S	voiume:	22230.40
П	Length: Width: Height:	12.55 Calculate
W	Area: Volume:	961.84
D		•

- 7. Save and close the form
- 8. Close Microsoft Access

<u>Previous</u>	Copyright © 2002-2004 FunctionX, Inc.	<u>Next</u>
-----------------	---------------------------------------	-------------


Introduction

Microsoft Access and Visual Basic ship with a lot of functions and procedures you can use in your database. Before creating your own procedures, you should know what is already available so you don't have to re-invent and waste a great deal of your time. The functions already created are very efficient and were tested in various scenarios so you can use them with complete reliability. The available functions range in various types. There are so many of them that we can only review those that you are most likely to use very often. You can find out about the other in the Help files because they are fairly documented.

Conversion Functions

Anything the user types in a text-based field is primarily considered as a string. Although Microsoft Access and Visual Basic perform a great deal of work behind the scenes to validate the various values that the user provides during data entry, sometimes you will need to make sure that a value is of the type you are expected. This is also valid even if the user did not enter a certain value, you would still need to retrieve a value and convert it to the right data type.

There are various conversion functions adapted to the data types they are dealing with. The general syntax of the conversion functions is:

ReturnType = Function(Expression)

The expression could be of any kind, depending on how the expression would be supplied. For example, it could be a string or value the user would have entered in form. It could also be the result of a calculation performed from other fields on the database. The function would take such a value, string, or expression and attempt to convert. If the conversion is successful, the function would return a new value that is of the type specified by the *ReturnType* in our syntax.

The	conversion	functions	are as	follows:	

Fund	ction	
Name	Return Type	Description
CBool	Boolean	Converts an expression into a Boolean value
CByte	Byte	Converts an expression into Byte number
CDate	Date	Converts and expression into a date or time value
CDbl	Double	Converts an expression into a flowing-point (decimal) number
CInt	Integer	Converts an expression into an integer (natural) number
CCur	Currency	Converts an expression into a currency (monetary) value

CLng	Long	Converts an expression into a long integer (a large natural) number
CSng	Single	Converts an expression into a flowing-point (decimal) number
CStr	String	Converts an expression into a string

Creating a Message Box

A message box is a special dialog box used to display a piece of information to the user. As opposed to a regular form, the user cannot type anything on the dialog box. There are usually two kinds of message boxes you will create: one that simply display information and one that expects the user to make a decision.

There are two methods you can use to create a message box using the **MsgBox** function. To display a simple message with just an OK button, use the **MsgBox** method whose formula is

MsgBox [Message]

Ň

On the other hand, you can ask the message box to display a more informative prompt with more than one button. Then you can write an expression that would perform an action based on the particular button the user clicked. In this case, you would have to use the MsgBox function (not the method).

The syntax of the MsgBox function is

```
MsgBox [Message] [Buttons] [Title] [HelpFile] [Context]
```

The **MsgBox** method takes only one argument. The **MsgBox** function takes five arguments. The *Message* argument is required on both.

The *Message* argument is the string that the user will see displaying on the message box. As a string, you can display it in double quotes, like this "That's All Folks". You can also create it from other pieces of strings. The Message argument can be made of up to 1024 characters. To display the *Message* on multiple lines, you can use either the constant **vbCrLf** or the combination **Chr(10) & Chr(13)** between any two strings.

The *Buttons* argument specifies what button or buttons should display on the message box. There are different kinds of buttons available and Visual Basic recognizes them by a numeric value assigned to each.

Button	Value	Display				
vbOKOnly	0	ОК				
vbOKCancel	1	0	К	Car	ncel	
vbAbortRetryIgnore	2	Abort	Ret	ry	Ign	ore
	_					

The Buttons argument can have one of the following constant

vbYesNoCancel	3	Yes	;	N	lo	Ca	ncel
vbYesNo	4		Ye	es	N	lo	
vbRetryCancel	5		Re	try	Car	ncel	

From the buttons on the message box, you can decide which one would be the default, that is, which button would be activated if the user presses Enter instead of clicking. You can set the default argument using the following table

Option	Value
vbDefaultButton1	0
vbDefaultButton2	256
vbDefaultButton3	512
vbDefaultButton4	768

These additional buttons can be used to further control what the user can do:

Constant	Value
vbApplicationModal	0
vbSystemModal	4096

In addition to the buttons, you can use the following icons on the message box

Icon	Value	Description
vbCritical	16	8
vbQuestion	32	ৃ
vbExclamation	48	<u>•</u>
vbInformation	64	•

The *Title* argument is the caption that would display on the title bar of the message box. It is a string whose word or words you can enclose between parentheses or that you can get from a created string.

If your application is using a help file, you can specify this and let the message box use it. The *HelpFile* argument is a string that specifies the name of the help file, and the Context argument provides the number that corresponds to the appropriate help topic for the message box.

As a function, the **MsgBox** function is useful for the value it returns. This value corresponds to the button the user clicks on the message box. Depending on the buttons the message box is displaying, after the user has clicked, the **MsgBox** function can return one of the following values:

Button			Return	Value
	ОК		vbOK	1
	Cancel		vbCancel	2
	Abort		vbAbort	3
	Retry		vbRetry	4
	Ignore		vbIgnore	5
	Yes		vbYes	6
	No		vbNo	7

Practical Learning: Creating Message Boxes

- 1. Open the VBAccess1 application and, on the Database Window, click Forms
- 2. Double-click the Messages form to open it

🖀 Messages to the People 🔀				
Employee Name:				
Message <u>1</u>	Message <u>5</u>			
Message <u>2</u>	Message <u>6</u>			
Message <u>3</u>	Message <u>7</u>			
Message <u>4</u>	Message <u>8</u>			

- 3. Switch the form to Design View.
- 4. Right-click the Message 1 button and click Build Event... On the Choose Builder dialog box, click Code Builder and click OK
- 5. In the Code Editor, implement it as follows:

```
Private Sub cmdMessage1_Click()
    MsgBox "This is Visual Basic as simple as it can get"
End Sub
```

- 6. To test the form, return to Microsoft Access and switch the form to Form View.
- 7. On the form, click the Message 1 button.
- 8. Notice that a message box displays. Also notice the caption on the title bar displays Microsoft Access.
- 9. Click OK to close the message box.
- 10. Switch the form back to Design View and return to the Code Editor
- Instead of the title bar displaying Microsoft Access as the caption, you can set your own caption. This is done through the 3rd argument of the **MsgBox** function. To see an example, on the Object combo box, select cmdMessage2 and implement its Click event as follows:

```
Private Sub cmdMessage2_Click()
    MsgBox "Before formatting a floppy disk, " & _
    "make sure you know its content", , _
    "Disk Formatting Instructions"
End Sub
```

Lesson 6: Built-In Functions and Procedures

- 12. Test the form and the Message 2 button. Then return to the Code Editor
- 13. When creating a message box using the MsgBox function, you can decide which button you want to use, using one of the constants we have listed earlier. To see an example, on the Object combo box, select cmdMessage3 and implement its Click event as follows:

```
Private Sub cmdMessage3_Click()
    MsgBox "This will be your only warning", _
    vbOKOnly + vbExclamation, _
    "Attention! Attention!! Attention!!!"
End Sub
```

- 14. Test the form and the Message 3 button. Return to the Code Editor
- If you want to display a message on different lines, you can use the vbCrLf constant. As an example, on the Object combo box, select cmdMessage4 and implement its Click event as follows:

```
Private Sub cmdMessage4_Click()

MsgBox "You are about to embark on a long journey." & _
vbCrLf & "If your courage is still fresh, " & _
"now is the time to let us know!", _
vbOKCancel + vbQuestion, _
"Accept or Cancel the Mission"
End Sub
```

- 16. Test the form and experiment with the Message 4 button. Then return to the Code Editor.
- You can also display a message on various lines using the Chr() function. To see an example, on the Object combo box, select cmdMessage5 and implement its Click event as follows:

```
Private Sub cmdMessage5_Click()

MsgBox "This message usually appears when trying " & _
    "to format a floppy disk while the floppy drive " & _
    "is empty. " & Chr(13) & Chr(10) & _
    "When or if this happens, make sure you have a " & _
    "floppy disk in the floppy drive.", _
    vbAbortRetryIgnore + vbCritical, _
    "Floppy Disk Formatting"
End Sub
```

- 18. Test the form and the Message 5 button. Then return to the Code Editor.
- 19. The usefulness of the **MsgBox** function is demonstrated in your ability to perform an action based on the button the user has clicked on the message box. Indeed, the implementations we have used so far were on the MsgBox method. If you want to get the button that the user has clicked, you have to use the function itself. The true capture of the clicked button is revealed by your finding out the clicked button. This is done using conditional statements that we have not learned so far. Therefore, we will just learn how to implement the function and how to assign a response button to it; throughout this tutorial, and whenever

necessary, we will eventually see what to do when a certain button has been clicked. To see an example, on the Object combo box, select cmdMessage6 and implement its Click event as follows:

```
Private Sub cmdMessage6 Click()
   Dim intResponse As Integer
    intResponse = MsgBox("Your printing configuration " &
                         "is not fully set." & vbCrLf &
                         "If you are ready to print, click" & vbCrLf &
                         "(1) Yes: To print the document anyway" & vbCrLf &
                         "(2) No: To configure printing" & vbCrLf &
                         "(3) Cancel: To dismiss printing", _
                         vbYesNoCancel + vbInformation,
                         "Critical Information")
End Sub
```

20. Test the form and the Message 6 button. Then return to the Code Editor

21. When a message box displays, one of the buttons, if more than one is displaying, has a thicker border than the other(s); such a button is called the default button. By default, this is the 1st or most left button on the message box. If you want to control which button would be the default, use one of the default constant buttons listed above. To see an example, on the Object combo box, select cmdMessage7 and implement its Click event as follows:

```
Private Sub cmdMessage7 Click()
  Dim intAnswer As Integer
  "operation?",
                vbYesNoCancel + vbQuestion + vbDefaultButton2,
                "National Inquiry")
```

End Sub

- 22. Test the form and the Message 7 button. Then return to the Code Editor.
- 23. Although the user cannot type on a message box, not only can you decide what it displays, but you can also use string variables that would be available only when the form is running. As an example, on the Object combo box, select cmdMessage8 and implement its Click event as follows:

```
Private Sub cmdMessage8 Click()
   Dim strEmplName As String
   Dim intInquiry As Integer
   strEmplName = CStr(txtEmployeeName)
   intInquiry = MsgBox("Hi, " & strEmplName & Chr(13) &
                        "I think we met already." & vbCrLf &
                        "I don't know when. I don't know where." & vbCrLf &
                       "I don't know why. But I bet we met somewhere.",
                       vbYesNo + vbInformation,
                        "Meeting Acquaintances")
```

End Sub

- 24. To test the form, return to Microsoft Access
- 25. On the Employee Name text box, type Joseph Douglas
- 26. Click the Message 8 button and see the result.
- 27. Click one of the buttons to close the message box.
- 28. Close the running form, click its close button \blacksquare .

Data Request With the Input Box

Although most of the user's data entry will be performed on fields positioned on a form, Microsoft Visual Basic provides a function that allows you to request a piece of information to the user who can type it in a text field of a dialog box. The function used to accomplish this is called InputBox and its basic syntax is:

InputBox(prompt)

The most basic piece of information you should provide to the InputBox function is referred to as the prompt. It should be a string that the user will read and know what you are expecting. Here is an example:

```
Private Sub cmdRequestDOB_Click()
    InputBox "Enter your date of birth as mm/dd/yyyy"
End Sub
```

This would produce



Upon reading the message on the Input box, the user is asked to enter a piece of information. The type of information the user is supposed to provide depends on you, the programmer. Therefore, there are two important things you should always do. First you should let the user know what type of information is requested. Is it a number (what type of number)? Is it a string (such as the name of a country or a customer's name)? Is it the location of a file (such as C:\Program Files\Homework)? Are you expecting a Yes/No True/False type of answer (if so how should the user provide it)? Is it a date (if it is a date, what format is the user supposed to enter)? These questions can lead you to believe that you should state a clear request to the user and specify what kind of value you are expecting. For example, instead of the question above, you can implement the InputBox function as follows: Lesson 6: Built-In Functions and Procedures

```
Private Sub cmdRequestDOB_Click()
    InputBox "Please enter your date of birth as mm/dd/yyyy"
End Sub
```

Another solution, also explicit enough, consists of providing an example to the user.

The second thing you should take care of is the value the user would have typed. After typing a value, the user would click one of the buttons: OK or Cancel. If the user clicks OK, you should retrieve the value the user would have typed. It is also your responsibility to find out whether the user typed a valid value. Because the **InputBox()** function can return any type of value, it has no mechanism of validating the user's entry. To retrieve the value of the Input Box dialog when the user clicks OK, you must use the **InputBox()** function (and not the method as above). Here is an example:

```
Private Sub cmdRequestDOB_Click()
    Dim dteDOB As Date

    dteDOB = InputBox("Please enter your date of birth as mm/dd/yyyy")
    txtDOB = dteDOB
End Sub
```

Sometimes, even if you provide an explicit Input Box, the user might not provide a new value but click OK. The problem is that you would still need to get the value of the text box and you might want to involve it in an expression. You can solve this problem and that of providing an example to the user by filling the text box with a default value. Besides the prompt, Visual Basic provides a more elaborate InputBox function that allows you to specify more options, including a default value. The syntax used then is:

InputBox(prompt, Title, Default, XPos, YPos, HelpFile, Context)

Using this syntax, you can provide a title to display on the title bar of the Input Box dialog. This is taken care of by the Title. The XPos and YPos arguments allow you decide the position of the Input Box from left (XPos) and top (YPos) measurements of the screen.

Boolean Functions: IsEmpty

The **IsEmpty** function states that a field is empty. The syntax of the **IsEmpty** function is

IsEmpty(Expression)

In this case, the Expression argument will be set as empty. The **IsEmpty** function returns a Boolean value. The **IsEmpty** function is mostly used to check whether a field is empty. Then you would use the result for another purpose. The result in this case would be expressed as TRUE or FALSE.

Example =IsEmpty(MI)

This would result in the MI field being empty

Boolean Functions: IsNull

The **IsNull** function states an expression as being Null, which means the expression is set as a field that contains no value. The syntax of the **IsNull** function is

IsNull(*Expression*)

The **IsNull** function returns a Boolean value.

Previous Copyright © 2002-2004 FunctionX, Inc. Next

Properties of Forms, Reports, and Controls

Introduction

Users interact with your database by using forms and the controls those forms host. As a database developer, you create those forms and decide what controls are necessary to make your database a good product. To make the forms of your database as useful as possible, you should know what the various controls provide, which is primarily their properties.

A property is something that characterizes or describes an object. A property can be the name of an object, its color, the text it displays, or its width.

Controls Properties

When designing a form and manipulating <u>Windows controls</u>, you will use a central object called the Properties Window:

····	
E Form	
Format Data Event C	Other All
Record Source	Clients
Filter	
Order By	
Allow Filters	Yes
Caption	Clients
Default View	Single Form
Views Allowed	Both
Allow Edits	Yes
Allow Deletions	Yes

Made of five property sheets, the Properties window displays the properties associated with the object or the control that is selected on the form. To get the Properties window of the properties associated with a control, right-click that control and click Properties.

To display the Properties window for the form, double-click the button that is at the intersection

of both rulers . You can also click the Properties button on the Form Design toolbar to toggle the Properties windows.

-8	Fo	orr	n1	:	Fo	rn	n													9			Ľ.					I	_			<			
	I	•	•	•	I	•		•	1	•	•	•	I	•	•	•	2	•	•	· I	•	•	•	3	•	•	•	I	•	•					
	IГ	Ŧ	De	ta	il		55			9							88																		
ŀ														E	T	Fo	rm	1																	×
11															Fo	rma	at	1	Da	ta	1	E٧	eni	t		Oth	er			Al	I				
1								 							Re	всо	rd	So	uro	е.													 		
ŀ															Fi	lter rde	r F		• •	• •	•••	• •	8-										 	 -	
:															A	low	Fi	ilte	rs .				Y	'es											
						ï	••••	 	1						C	apt	іоп	۱.															 		
	-														De	efa	ult	Vie	BW	•••	• •	•••	S	ing	jle	Fo	rm						 		
															Vi	ew:	s A	llo	wei	d .	• •	•••	B	loth	h								 	 -866	
															A	low	E	dits	5.	• • •		• •	Y	'es										-88	
															A	Iow	D I	ele	tio	ns .	• • •	• •	Y	'es									 	 -	
															AI	IOW	/ A	ddi	itioi	ns .	•••	• •	Y	'es									 	 -	
															Da	ata	Er	ntry	1.	• •	• • •	• •	N.	lo									 		
															Re	eco	rd:	set	Ту	pe	• •	• •	D)yn	as	et							 		
															Re	eco	rd	Lo	cks				N	lo l	LO	:ks							 		

The properties of an object are divided in three main categories. Characteristics that control the form's appearance are listed in the **Format** property sheet. Characteristics that control internal or external relationships, as far as data is concerned, are listed in the **Data** property sheet. The **Other** property sheet lists characteristics that don't fit in one of the other groups. When dealing with a particular characteristic, you can access it from one of those property sheets or you can use the whole list of properties from the **All** property sheet.

V Practical Learning: Viewing the Form's Properties Window

- 1. Open the CPAP Help Desk1 database and click the Forms button or property sheet
- 2. In the Forms section, Right-click Suppliers and click Design.
- To display the Properties window, double-click the button at the intersection of the rulers
 Notice that the Properties window displays 5 property sheets

4. To switch the form to Form View, on the Form Design toolbar, click the View button.

5. To close the form, click its close button \blacksquare .

Properties Categories

Properties are set by changing their values and there are various types of properties.

In the Properties window of Microsoft Access, a property is made of two parts: its name and its value:

	😭 Form		×
Property's	Format Data Eve	nt Other All	
Name	Record Source		
1	Filter		
	Order By		
Property's	Allow Filters	Yes	
Value	Caption		
	Default View	Single Form	
	Views Allowed	Both	
	Allow Edits	Yes	
	Allow Deletions	Yes	
	Allow Additions	Yes	
	Data Entry	No	
	Recordset Type	Dynaset	
	Record Locks	No Locks	•

The name of a property displays on the left gray section. Although you can click it to select it, you cannot change it. The property name can be made of one word such as **Width**; it can also be made of a combination of words, such as **Record Source**. On this site, a property will be called by what displays on that left gray section. This means that, if a property displays "Height", we will call it "The Height Property". If it displays "Whats This Button", we will call it "The Whats This Button Property".

In Visual Basic, the name of a property is made of one word starting in uppercase. Examples are **Text** or **Caption**. If the property is a combination of words, each component of the word would start in uppercase. Examples are **RecordSource** or **AllowAdditions**. Not all properties of the Microsoft Access' Properties window are exactly reproduced in Visual Basic but most of them are. And most of the properties have the same name, in one word.

Before referring to a property in Visual Basic, you must first call the control whose property you want to use. There are various ways you can call a control. If you know the name of a control, you can type it and assign it default value. Here is an example:

```
Private Sub Form_Load()
    txtFirstName
End Sub
```

Alternatively, you can let Visual Basic know that you are referring to an object that belongs to the same form in which the event is being written. In Visual Basic, a form refers to itself as **Me**. In the Code Editor, after typing **Me**, type the period operator. Visual Basic would display the list of properties and the controls that the form is hosting. This allows you to select the control or property you need. Using the Me object, you can access a control:

```
Private Sub Form_Load()
    txtFirstName
    Me.txtLastName
End Sub
```

In the Properties window, the second part of a property displays on the right side of the property name. This is called the value of the property:

Lesson 7: Properties of Forms, Reports, and Controls

	🖆 Form	×
Property's	Format Data Eve	ent Other All
Name 🗸	Record Source	
1	Filter	
	Order By	
Property's	Allow Filters	Yes
Value	Caption	
	Default View	Single Form
	Views Allowed	Both
	Allow Edits	Yes
	Allow Deletions	Yes
	Allow Additions	Yes
	Data Entry	No
	Recordset Type	Dynaset
	Record Locks	No Locks

As properties are intended to fulfill different purposes, there are various categories of properties. In the Properties window, to give a value to a property, you type the desired value in the field on the right side of the name of you proceed as we will learn shortly, depending on the property type.

In Visual Basic, to change the value of a property, you can assign it the desired value, using the assignment operator

String Properties

Record Source	Employees
Filter	
Order By	

Order By..... In the Properties window, some properties display a word or a group of words. Such a word or group of words on a field is called a string. Some of those properties already display a value while some others are empty. To set or change such a string, you can click the property name and type the desired value, which would replace whatever text the property was displaying. Alternatively, you can edit the text that the property already has.

There are various types of string properties on the form and other controls. Examples include among others, the control's name or its caption.

To provide a string value to a property in Visual Basic, use the assignment operator and type the text in double-quotes. Here is an example:

```
Private Sub Form_Load()
    Me.Caption = "Visual Basic for Applications"
End Sub
```

String Properties: Controls Names

Everything, that is every object, in your database needs a name (in fact, this includes every object used by the computer). A name allows the database engine (and the computer) to identify each object for any necessary reference. Although the name of a table (query, form, or report) doesn't appear as obvious as some other controls, as a property, it is highly valuable when creating expressions that involve the table (query, form, or report).

There are two main ways an object (in this section, the word object includes not only the form but other Windows controls that we will be using during form and report designs) receives a name. If you create a form using the New Object button on the Table Datasheet toolbar, by default, Microsoft Access gives it the same name as the form (or the query) where its data originates from. Also, when you create a form using the Form Wizard, Microsoft Access would suggest the name of the source table (or query) to the form.

In these last two examples, you can accept the suggested name. Because a form and a table are two distinct objects, they can have the same name without any conflict. As an example, if you create a database using the Database Wizard, you would see that the tables and forms hold the same names.

The second option you have is to change the suggested name of the form. As we did with the tables, to set the name of a form, if it has not been previous saved (by you), you must ask to save the form and give the name you want. The name of a form follows the same rules we applied to tables. In this case, to identify a form as such, you can start its name with **frm** followed by a word or group of words that can easily identify what the form is being used for.

Practical Learning: Setting or Changing a Form's Name

- 1. In the Forms section, right-click Suppliers and click Copy
- 2. Right-click an unoccupied area in the Forms section and click Paste
- 3. In the Paste As dialog box, type **frmSuppliers** and press Enter twice (the first Enter register the name, the second Enter opens the form).

String Properties: The Caption

The most upper characteristic that appear on a form is the word or group of words on the title bar. This is called the caption of the form and it is controlled by the Caption property. By default, the title bar displays the name of the form. To change this title, access the **Caption** field in the **Format** property sheet and type the desired title. Just like the caption of a table, the title of a form is not related to the form's name; therefore, you can type any caption you want, but a caption should reflect what the form is being used for.

Practical Learning: Setting the Form's Caption

- 1. To switch the form to Design View, on the Form View toolbar, click the View button.
- 2. To change the caption of the form, on the Properties window, click Format
- 3. The cursor should be blinking in the Caption field; if not, click **Caption**. Type **College Park Auto-Parts - Supplies Records**
- 4. To view the caption, you must switch to Form View. On the menu bar, click View -> Form View.
- 5. After viewing the caption, on the menu bar, click View -> Design View to switch the form to design mode.

Boolean Fields

Allow Filters	Yes
Allow Edits	Yes
Allow Deletions	Yes
Allow Additions	Yes
Data Fotry	No

Data Entry No Some fields behave like a True or False answer to a question. Such fields can display only one of two values. Both values come in a combo box that displays as the property field. There are three categories of these fields in combinations of Yes/No, True/False, or On/Off.

There are two main ways you use a Boolean field: either to set its value or to find out what value the field is holding. When studying conditional statements, we will learn the mechanics of finding out what value a property is holding. To change the value of the field in the Properties window, click the field to display its combo box. Then click the arrow of the combo box and select the other value.

To change the value of a Boolean field or variable in Visual Basic, assign it a True or False value, depending on your goals. Because the Code Editor would not display the list of possible values on the right side of the assignment operator, you can press Ctrl + Space to show the list.

From the list, you can select and double-click True or False. Of course, it is faster and easier to type the desired value since you already know what it should be.

If you are invoking a property of the Properties window, it is likely that Visual Basic is already aware that the property is Boolean. In that case, after typing the assignment operator, Visual Basic would display both True and False in a two-item list: Lesson 7: Properties of Forms, Reports, and Controls

***	Form_Form1 : Class Module 📃 🗖 🔀
cr	ndGetMarried Click
	Option Compare Database Option Explicit
	Private Sub cmdGetMarried_Click() Me.NavigationButtons = End Sub
	Private Sub Form_Load() Me.Caption = "Visual Basic for Applications" End Sub
=	

You can type t or f and press Tab (or Enter) to select the desired value.

Navigating Buttons

■ ■ ■ When studying tables and during form's data entry, we found out that a form (also a table, a query, or a report) is equipped with some buttons on its lower section. These buttons allow the user to navigate back and forth through records. These buttons are very useful during data entry and data analysis. If you are creating a form that would display the same content all the time, such as a form that doesn't actually display records, you can hide the form navigation buttons.

The presence or absence of navigation buttons is controlled by the Boolean **Navigation Buttons** property . When its value is set to **Yes**, the form would display the Navigation Buttons. To hide them, set the Navigation Buttons property value to **No**.

Practical Learning: Hiding the Navigation Buttons

- 1. On the Form Design toolbar, click the Code button.
- 2. In the Object combo box, click Form and implement its Load event as follows:

```
Private Sub Form_Load()
    Me.NavigationButtons = False
End Sub
```

- 3. Return to the form and switch it to Form View.
- 4. Notice that the form doesn't have the bottom navigation buttons. After using the form, switch it to Design View and get back to the Code Editor

Enumerated Properties

Lesson 7: Properties of Forms, Reports, and Controls

Picture Type	Embedded
Picture Size Mode	Clip
Picture Alignment	Cepter

Picture Alignment Center A set is a list of words that composes the possible values of a property. The list, which is static, which means it cannot be changed, comes as a combo box where the user (in this case the user is the database developer) can select one item from. To change the property value in the Properties window, click the arrow of the combo box and select the desired value from the list.

To set or change the value of an enumerated property in Visual Basic, you must know the value you want. There are two main alternatives: using your experience or getting help. If you know the possible values of the property, you can assign the desired value to the property name. If you don't know the possible values, you can get back to the Properties window, click the arrow of the combo box that is the property whose value you want to assign, view and write down the possible values. Here is an example:

😭 Form	ı		
Format	Data Eve	nt Other All	
Width .		5"	~
Picture .		(none)	_
Picture 1	Гуре	Embedded	
Picture S	Size Mode	Clip	
Picture #	Alignment	Center 🗾	
Picture 1	Filing	Top Left	
Grid X .		Top Right	
Grid Y .		Center	
Layout f	or Print	Bottom Left	
Palette S	5ource	Bottom Right	
		Form Center	

Once you have the list of possible values, you can get to the Code Editor and get ready to assign the desired value. Many of the values of properties from the Properties window are not exactly equivalent in Visual Basic. This means that you may not use the same values as in the Properties window. The help files can give you the list of possible values.

Numeric Properties

Left	1.25"
Тор	0.25"
Width	1.6042"
Height	0.1771"

Height 0.1771" Some properties use a value that is a number. There are two types of numbers you will be asked to provide, depending on the property. An integer is a number that doesn't take a decimal portion. Such a number can be made of digits only. For such a field, make sure you provide an integer of appropriate range, as you will possibly be directed to do. The other type of number will be decimal.

A decimal number, also called a floating number, can be made of digits or a combination of digits and one period in between. When setting such a value, make sure that either you type only digits, or you type digits and one period; the period can be anywhere in the value, Microsoft Access would take care of formatting it if it judges it necessary.

To change the value of a numeric property in the Properties window, you can click the property name to highlight the property value. Then you can type the desired value.

To set the numeric value of a property is Visual Basic, use the assignment operator and assign the desired value to the property. Once again, the value must be a valid number.

Practical Learning: Setting Numeric Properties

1. In the Code Editor, change the Load event of the form as follows:

```
Private Sub Form_Load()
    Me.NavigationButtons = False
    Width = 4.128
End Sub
```

- 2. Get back to the form and switch it to Form View to preview it. On the main menu, click Window -> Size to Fit Form
- 3. Switch the form back to Design View.

Expressions Properties

Input Mask..... Default Value Validation Rule Validation Rule Some properties display a value that is a more elaborate expression. If you know what value the property should display, namely an expression, you can just type it. Otherwise, Microsoft Access provides intermediary steps that you can follow to create or build the value. We will see examples of those when we study expressions and queries.

Using the From Wizard

The Form Wizard provides the easiest means of creating a form automatically. Like the objects wizards we had seen so far, this one also takes you step by step through creating the object.

There are two main ways you can launch the Form Wizard: from the Database Window or from the New Form dialog box. On the Database Window, click the Forms button. Then double-click Create Form By Using Wizard. On the New Form dialog box, click Form Wizard and click OK.

The first page of the Form Wizard allows you to choose the originating table or query that will supply the necessary fields in the form. Once you have selected the object, its corresponding fields display in the Available Fields list box, you can select all fields or decide which ones you want to include in the form.

The 2nd page of the wizard allows you to select the desired layout of the form. Forms can be designed in various flavors: Columnar, Datasheet, Tabular, Justified, or a mix:

 A columnar form is used to display data one record at a time. This is a convenient display for data entry and analysis because the user is able to examine each piece of information for each one record:

88	Expense Codes	
I		
	Expense Code ID	1
	Expense Code	Office Supplies
Re	cord: 🚺	1 🕨 🕨 🕨 of 1

• A tabular form displays its data in a table layout following a continuous series of records. All or almost all records are displaying in a single layout:

-8	Music Collecti	on - Albums Tracks				×
	Track ID	Album Title	#	Track Title	Length	
	9	Tribute	9	Waltz In 7/8	5:32	
	10	Tribute	10	Nightingale	5:44	
	11	Tribute	11	Niki Nana (We're One)	8:15	
	12	Dangerous	1	ЈАМ		
	13	Dangerous	2	Why You Wanna Trip On		
Re	cord: I	<u> </u>	of 48			

There are three main ways you can create a tabular form: from the New Form dialog box, using the Form Wizard, or designing it.

• A datasheet form looks and behaves like a table, displaying all possible records at the same time instead of one record at a time:

-8	Depreciation: N	et Value			×
	DepreciationIE	AssetID	Depreci	DepreciationArr	
►	1	1	/31/1994	\$40.00	
	2	1	/28/1994	\$40.00	
	3	1	/31/1994	\$40.00	
	4	2	/28/1994	\$50.00	
	5	2	/31/1994	\$50.00	
	6	3	/31/1994	\$7.00	
	7	3	/28/1994	\$7.00	
	8	3	/31/1994	\$7.00	
	9	5	/31/1994	\$15.00	-
Re	cord: 🚺 🔳	1	▶ ▶1 ▶*	f of 9	1.

A datasheet is mainly used in relationships to display another table's related records. It is also suitable for people who prefer to work in a spreadsheet environment. This display allows the database developer to provide a sheet view to the data entry personnel without making the table's design or formats available.

You can make the same form available in Form View and Datasheet View. In fact, most forms are. The users can switch from Form View to Datasheet View by clicking the View menu. Unfortunately, this could also allow the users to get the form in Design View, then they could modify it. If you don't want the users to have access to Design View but retain Form View and Datasheet View, you might have to create custom menus and toolbars.

There are two main ways you can create a Datasheet form: using the wizard or designing it.

• A Justified form provides a good and consistent look. When the form is created, borders are added to labels:

🗉 The Associates - Assets								
	Asset ID					-		
	Asset Description				1			
	Personal Computer							
	Davolio, Nancy 🗸	Computer -		Sold 🚽				
	Model Number	Serial Number		Barcode#				
		12344111						
	Date Acquired Date Sold Pur	chase Price	Depreciation Method Depreciable Li		Depreciable Life			
	1/1/1994	\$2,500.00	SL		5			
	Salvage Value Current V							
	\$100.00				\$1,400.00	-		
Record: 1 1 1 1 1 1 6 5								

The 3rd page of the Form Wizard presents the forms designs you can choose from. These are the designs we saw when creating a database using the Database Wizard.

The 4th page allows you to name the form, using the naming convention we studied when creating the tables by providing an appropriate name.

Previous Copyright © 2002-2004 FunctionX, Inc. Next	<u>Previous</u>	Copyright © 2002-2004 FunctionX, Inc.	<u>Next</u>
---	-----------------	---------------------------------------	-------------



Conditional Statements

Introduction

The essence of computer programming is on telling the computer what to do when something occurs, and how to do it. This is performed by setting conditions, examining them and stating what decisions the computer should make.

To perform the necessary conditions, you have two main options: Microsoft Access or Microsoft Visual Basic. Microsoft Access is equipped with a series of operators and functions destined to perform various operations. To use a condition in Microsoft Access, if you know the structure of the conditional statement, you can write it after typing an assignment operator. Because most conditions in Microsoft Access (namely IIf) are in the form of functions, we will study them later on. For now, we will learn how to write conditions in Visual Basic.

Microsoft Visual Basic comes with a lot of conditional statements for almost any situation your computer can encounter. As the application developer, it is up to you to anticipate these situations and make your program act accordingly.

A comparison is performed between two values of the same type; for example, you can compare two numbers, two characters, or the names of two cities. On the other hand, a comparison between two disparate values doesn't bear any meaning. For example, it is difficult to compare a telephone number and somebody's grand-mother name, or a music category and the distance between two points. Like the arithmetic operations, the comparison operations are performed on two values. Unlike arithmetic operations where results are varied, a comparison produces only one of two results. The result can be a logical **True** or **False**.

To perform the necessary comparisons, Microsoft Access and Visual Basic use a series of logical operators and constants.

Logical Operators

The Comparison for Equality =

To compare two variables for equality, use the = operator. Its syntax is:

Value1 = Value2

The equality operation is used to find out whether two variables (or one variable and a constant) hold the same value. From our syntax, the value of Value1 would be compared with the value of Value2. If Value1 and Value2 hold the same value, the comparison produces a **True** result. If they are different, the comparison renders **false** or 0.



The Logical Not Operator

When a variable is declared and receives a value (this could be done through initialization or a change of value) in a program, it becomes alive. When a variable is not being used or is not available for processing (in visual programming, it would be considered as disabled) to make a variable (temporarily) unusable, you can nullify its value. To render a variable unavailable during the evolution of a program, apply the logical not operator which is **Not**. Its syntax is:

Not Value

There are two main ways you can use the logical **Not** operator. As we will learn when studying conditional statements, the most classic way of using the logical **Not** operator is to check the state of a variable.

When a variable holds a value, it is "alive". To make it not available, you can "not" it. When a variable has been "notted", its logical value has changed. If the logical value was **True**, it would be changed to **False**. Therefore, you can inverse the logical value of a variable by "notting" or not "notting" it.

For Inequality

As opposed to Equality, Visual Basic provides another operator used to compare two values for inequality, which is <>. Its syntax is:

Value1 <> Value2

The <> is a binary operator (like all logical operators except the logical **Not**, which is a unary operator) that is used to compare two values. The values can come from two variables as in Variable1 <> Variable2. Upon comparing the values, if both variables hold different values, the comparison produces a **True** value. Otherwise, the comparison renders **False** or a null value.



The inequality is obviously the opposite of the equality.

A Lower Value <

To find out whether one value is lower than another, use the < operator. Its syntax is:

Value1 < Value2

The value held by Value1 is compared to that of Value2. As it would be done with other operations, the comparison can be made between two variables, as in Variable1 < Variable2. If the value held by Variable1 is lower than that of Variable2, the comparison produces a **True**.



Combining Equality and Lower Value <=

The previous two operations can be combined to compare two values. This allows you to know if two values are the same or if the first is less than the second. The operator used is \leq and its syntax is:

Value1 <= Value2

The <= operation performs a comparison as any of the last two. If both Value1 and VBalue2 hold the same value, result is true or positive. If the left operand, in this case *Value1*, holds a value lower than the second operand, in this case *Value2*, the result is still true.



A Greater Value >

When two values of the same type are distinct, one of them is usually higher than the other. Visual Basic provides a logical operator that allows you to find out if one of two values is greater than the other. The operator used for this operation uses the > symbol. Its syntax is:

Value1 > Value2

Both operands, in this case Value1 and Value2, can be variables or the left operand can be a variable while the right operand is a constant. If the value on the left of the > operator is greater than the value on the right side or a constant, the comparison produces a **True** value. Otherwise, the comparison renders False or null.



Greater or Equal Value >=

The greater than or the equality operators can be combined to produce an operator as follows: >=. This is the "greater than or equal to" operator. Its syntax is:

Value1 >= Value2

A comparison is performed on both operands: Value1 and Value2. If the value of Value1 and that of Value2 are the same, the comparison produces a **True** value. If the value of the left operand is greater than that of the right operand, the comparison still produces **True**. If the value of the left operand is strictly less than the value of the right operand, the comparison produces a **False** result.



Here is a summary table of the logical operators we have studied:

Operator	Meaning	Example	Opposite	
=	Equality to	a = b	<>	
<>	Not equal to	12 != 7	=	
<	Less than	25 < 84	>=	
<=	Less than or equal to	Cab <= Tab	>	
>	Greater than	248 > 55	<=	
>=	Greater than or equal to	Val1 >= Val2	<	

The If...Then Statement

The **If...Then** statement examines the truthfulness of an expression. Structurally, its syntax is:

If Condition Then Statement

Therefore, the program will examine the *Condition*. This condition can be a simple expression or a combination of expressions. If the *Condition* is true, then the program will execute the *Statement*



There are two ways you can use the **If...Then** statement. If the conditional expression is short enough, you can write it on one line using the following syntax:

If Condition Then Statement

In the following example, if the Gender text box of a form displays Male, the background color of the Detail section would be colored in light blue:

```
Private Sub Form_Current()
    If Gender = "Male" Then Detail.BackColor = 16772055
End Sub
```

If there are many statements to execute as a truthful result of the condition, you should write the statements on subsequent lines. Of course, you can use this technique even if the condition you are examining is short. If then you use the *Statement* on a different line, you must terminate the conditional statement with **End If**. The syntax used is:

```
If Condition Then
Statement
End If
```

The example above can be re-written as follows:

```
Private Sub Form_Current()
    If Gender = "Male" Then
        Detail.BackColor = 16772055
    End If
End Sub
```

If the condition needs to cover many lines of code, the syntax to apply is:

```
If Condition Then
Statement1
Statement2
Statement_n
End If
```

Lesson 8: Conditional Statements

Here is an example:

```
Private Sub Form_Current()
    If Gender = "Male" Then
        Detail.BackColor = 16772055
        FormHeader.BackColor = 16752478
        FormFooter.BackColor = 14511872
        End If
End Sub
```

The If...Then...Else Statement

The If...Then statement offers only one alternative: to act if the condition is true. Whenever you would like to apply an alternate expression in case the condition is false, use the **If...Then...Else** statement. The syntax of this statement is:

```
If ConditionIsTrue Then
Statement1
Else
Statement2
End If
```

The condition, in this *ConditionIsTrue*, would be examined. If it produces a true result, then the first statement, in this case *Statement1*, would be executed. If the condition (*ConditionIsTrue*) is false, the second statement, in this case *Statement2*, would be executed.

Here is an example:

```
Private Sub Form_Current()
    If Gender = "Male" Then
        Detail.BackColor = 16772055
    Else
        Detail.BackColor = 13034239
    End If
End Sub
```

If any of the expressions needs more than one line of code to have a complete implementation, you can include it in the needed section before the end of the section. Here is an example:

```
Private Sub Form_Current()
If Gender = "Male" Then
Detail.BackColor = 16772055
FormHeader.BackColor = 16752478
FormFooter.BackColor = 14511872
Else
```

Lesson 8: Conditional Statements

```
Detail.BackColor = 13034239
FormHeader.BackColor = 7452927
FormFooter.BackColor = 29670
End If
End Sub
```

The If...Then...ElseIf Statement

The **If...Then...ElseIf** statement acts like the **If...Then...Else**, except that it offers as many choices as necessary. The formula is:

If Condition1 Then Statement1 ElseIf Condition2 Then Statement2 ElseIf Condition_n Then Statement_n End If

The program will first examine the first condition, in this case *Condition1*. If *Condition1* is true, then the program would execute the first statement, in this case *Statment1*, and stop examining conditions. But if *Condition1* is false, then the program would examine *Condition2* and act accordingly. Whenever a condition is false, the program would continue examining the conditions until it finds one that is true. Once a true condition has been found and its statement executed, the program would terminate the conditional examination at End If.

The above syntax pre-supposes that at lease one of the conditions would produce a true result. Sometimes, regardless of how many conditions you use, it is possible that none of them would produce a true result. Therefore, in anticipation of such occurrence, you should provide an alternate statement that would embrace any condition that doesn't fit in the possible true results. This is done by combining an **If...Then...ElseIf** statements. The resulting syntax to use is:

```
If Condition1 Then
Statement1
ElseIf Condition2 Then
Statement2
ElseIf Condition3 Then
Statement3
Else
Statement_False
End If
```

In this case, if neither of the **If** and **ElseIf**s conditions was validated, then the last statement, in this case *Statement_False*, would execute.

The Select Case Statement

If you have a large number of conditions to examine, the **If...Then...Else** will go through each one of them, which could take long (although usually transparent to the user). Visual Basic offers an alternative of jumping to the statement that applies to the state of the condition. This is performed through the sue of the **Select Case** statement.

The syntax of the Select Case is:

```
Select Case Expression
Case Expression1
Statement1
Case Expression2
Statement2
Case Expression_n
Statement_n
End Select
```

Visual Basic would examine the *Expression* and evaluate it once to get a general result. Then it would compare the result of *Expression* with the *Expression* of each case. Once it finds one that matches, it would execute the corresponding *Statementn*.

Here is an example:

```
Private Sub cboMembership_AfterUpdate()
   Dim strMembership As String
   strMembership = [cboMembership].Text
   Select Case strMembership
      Case "Teen"
        txtPrice = "$25"
      Case "Adult"
        txtPrice = "$50"
      Case "Senior"
        txtPrice = "$35"
   End Select
End Sub
```

If you anticipate that there could be no match between the *Expression* and one of the *Expression*, you can use a **Case Else** statement at the end of the list. The statement would then look like this:

```
Select Case Expression
Case Expression1
Statement1
Case Expression2
```

Statement2 Case Expression3 Statement3 Case Else Statement_n End Select

Looping and Counting: Do...Loop

Loops are used to repeat an action. There are various variations of the **Do** loops.

The syntax of the **Do While** loop is:

Do While Condition Statement(s) **Loop**

The program will first test the *Condition*. If the *Condition* is true, the program would execute the *Statement* or *Statements* and go back to the **Do While** statement and test the condition again. This expression will execute the *Statement* or statements AS LONG AS the *Condition* is true, as many times as the *Condition* will be visited and found true. If the *Condition* is false, the program will skip the **Do While** statement and not execute any.

Since the **Do While** statement tests the *Condition* first before executing the *Statement*, sometimes you will want the program to execute the *Statement* first, then go back and test the *Condition*. Visual Basic offers a reverse to the syntax, which is:

Do

Statement(s) Loop While Condition

In this case, Visual Basic will execute the *Statement* or *Statements* first, then it will test the *Condition*. If the *Condition* is true, the program will execute the *Statement* again. The program will continue this examination-execution as long as the *Condition* is true. The big difference here is that even if the *Condition* is false, the program will have executed the *Condition* at least once.

An alternative to the **Do While** loop is the **Do Until** loop. Its syntax is:

Do Until Condition Statement(s)

Loop

This loop will first examine the *Condition*, instead of examining whether the *Condition* is true, it will test whether the *Condition* is false.

The other side of the **Do Until** loop would execute the *Statement* first, then it would examine the *Condition*. The syntax used is:

Do Statement(s) Loop Until Condition

Counting and Looping: For...Next

If you don't know how many times a statement needs to be executed, you can use one of the **Do** loops. But whenever you want to control how many times a statement should be executed, the **For...Next** loop offers a better alternative. The syntax used is:

For Counter = Start To End Statement(s) Next

Used for counting, the **For...Next** loop begins counting at the *Start* point. Then it examines whether the current value (after starting to count) is greater than *End*; if that's the case, the program exits the loop. It then executes the *Statement* or *Statements*. Next, it increments the value of *Counter* by 1 and examines the condition again. This process goes on until *Counter* = *End*.

The syntax above will increment the counting by 1 at the end of each statement. If you want to control how the incrementing processes, you can set your own, using the **Step** option. Here is the syntax you would use:

For Counter = Start To End Step Increment Statement(s) Next Counter

You can set the incrementing value to your choice. If the value of *Increment* is positive, the *Counter* will be added its value. This means that you can give it a negative value, in which case the *Counter* would be subtracted the set value.

Counting and Looping: For...Each

Since the **For...Next** loop is used to execute a group of statements based on the current result of the loop counting from *Start* to *End*, an alternative is to state various steps in the loop and execute a group of statements for each one of the elements in the group. This is mostly used when dealing with a collection of items.

The syntax used is:

For Each Element In Group Statement(s) Next Element The loop will execute the *Statement* or *Statement(s)* for each *Element* in the *Group*.

Previous Copyright © 2002-2004 FunctionX, Inc. Next



Fundamental Windows Controls

Introduction

To interact with the machine, the user of a computer uses objects called controls. The choice, appearance, and behavior of these controls is the responsibility of the application developer that you are. When creating an application, you should be aware of what your environment offers in terms of controls and what are the capabilities of those controls. Microsoft Access provides various controls you can use when designing your database application. If these controls are not enough, you can add some that ship with Visual Basic and sometimes you can purchase others from third-party vendors. To start, you should know what is already available in Microsoft Access and how you can expand over it.

The Form

The form is the most fundamental object of Microsoft Access and Visual Basic. It is a rectangular object whose main purpose is to host or carry other controls. This means that a form by itself is useless; it is the controls that it hosts that make its role significant on an application.

To create a form in Microsoft Access, if using Microsoft Access 97, open the New Form dialog box, select Design View and click OK. In Microsoft Access 2000 and 2002, a link on the Forms section of the Database Window, namely Create Form In Design View, allows you to jump to the Design View and create a form. The most basic thing to do is to save the form and voila, you have a form. We also know at this time that you can use the Form Wizard to easily create a completely configured form.

By default, the form you create in Microsoft Access is primarily used for data processing. For this reason, it is equipped with navigation buttons, a record selector bar and one or two dividing lines. A regular form in a Windows application doesn't need or use these three attributes. Therefore, if you don't need these visible traits on your form, you should set their Boolean properties to No.

Practical Learning: Creating a Form

- 1. Start Microsoft Access and open the **Bethesda Car Rental3** database.
- 2. On the Database Window, click Forms.
- On the main menu, click Insert -> Form. In the Design Form dialog box, make sure nothing is selected in the Table or Query combo box and double-click Design View
- 4. To save the form, on the Form Design toolbar, click the Save button \blacksquare .
- 5. Type **Switchboard** and press Enter

Fundamental Properties of a Form

As you can see, a form is not a very interactive object, but it displays a few characteristics that would make it very valuable and allow it to provide a unique role.

As far as the user is concerned, on top, a form has a title bar made of three sections. The system button on the left holds a menu that would allow the user to minimize, maximize, restore, move or close the form. On the right side of the system button, the title displays a word or a group of words that we know as the caption. This same section of the title bar, which is the main section of the title bar, also is equipped with a menu that allows you or the user to perform some of the actions necessary for a database form. On the right side, the title bar displays system buttons that would allow the user to minimize, maximize, restore or close the form. The characteristics of the title bar of a form are controlled by the following properties available from the Properties window of the form:

- The Control Box: When its Boolean property is set to Yes, the form can display the system icon on the left and the system buttons on the right. If you want neither the system icon nor the system buttons, you can set the Control Box property to No
- The Min Max Buttons: This is an enumerated property that controls the presence or not of the system buttons but it depends on the value of the Control Box property. If the Min Max Buttons property is set to None and the Control Box is set to Yes, the form would display the system icon and the system Close button. The user would not be able to minimize, maximize, or restore the form but he or she can still close the form. If the Min Max Buttons property is set to None and the Control Box property is set to No, the form would display neither the system icon nor any of the system buttons. Consequently, the user would not be able to close the form. This scenario can be used to control exactly how the form can be close the form. The Min Max Buttons property also allows you to hide the maximize button with Min Enabled or to hide the minimize button with Max Enabled or to display all three system button with the Both Enabled value
- The Close Button: If the Control Box property is set to No, we saw that the Min Max Buttons property can control which system button, either minimize or maximize, would be available. Since the Min Max Buttons property has no

bearing on the system Close button, you can use the Boolean Close Button property to decide whether the system Close button would be available to the user. The Close Button property doesn't show or hide the system Close button; it only enables or disables it.

If a form allows the user to close it, when the user closes a form, it fires an OnClose event. You can catch this to perform any necessary operation as a result of the form being closed.

The main area of the form can be referred to as its body. This is where the form would host other controls that would make the form useful. A form as a Windows object has distinguishable borders that make it resizable. This simply means that a user can widen, narrow, heighten, or shrink a form. The ability to resize a form is controlled by the Border Style property. For example, if you don't want the user to be able to resize the form, you can set its Border Style property to Thin. If you don't want borders at all, you can set this property to None. This also removes the title bar, which consequently prevents the user from minimizing, maximizing, restoring, moving, or closing the form.

Practical Learning: Setting Form's Properties

- 1. Using the All property sheet of the Properties window for the form, change its **Caption** to **Bethesda Car Rental**
- 2. Still using the Properties window for the form, set the **Record Selectors**, the **Navigation Buttons**, and the **Dividing Lines** properties to **No** each
- 3. Set the Border Style property to Thin
- 4. Save the form and switch it to Form View to preview it
- 5. Click anywhere on the Database Window. Notice that the current form can be positioned behind the Database Window
- 6. To bring the form back, on the main menu, click Window -> Bethesda Car Rental. Switch the form back to Design View

Form Design

As far as users are concerned, a form is a rectangular entity made for data Entry. Besides the form's characteristics that are visible to the user, behind the scenes, that is in Design View, a form can be made of three sections: the **Form Header**, the **Detail**, and the **Form Footer** sections. When all these three sections are not visible, if you need them, you can right-click anywhere on the form and click **Form Header/Footer**:



There is no perfect difference between these three sections. You can use (or not use) them as you wish. You can place any control on any section but it is traditional for database developers to place all data entry controls on the Detail section. Remember that the user would never be aware of these sections unless you use some mechanism to make them visible.

Each one of these sections is mostly characterized by its height and its visual characteristics.

Practical Learning: Using Form's Sections

- 1. Right-click in the form (under the Detail bar) and click Form Header/Footer
- 2. Double-click the Form Header bar.
- 3. In the Properties window, click Height and type .375
- 4. Double-click the **Form Footer** bar. On the Properties window, click Height and type **.3014**
- 5. Switch the form to Form View to preview it

Dialog Boxes
A dialog box is a rectangular object that is used to host or carry other controls:



A dialog box is primarily characterized by two main features: its title bar and its body. The title bar, on top of the dialog box, has a title and the system close button. Although this is the classic appearance of a dialog box, it is not strictly exclusive. Some dialog boxes display the system icon. On the right side of the title bar, a classic dialog box displays only the system Close button made of X. Again, this is not exclusive. It is not unusual for a dialog box to display the minimize and the maximize/restore buttons.

To use a dialog box, the user must open it one way or another. Your job is to decide how and when the user will be able to open a dialog box.

To create a dialog box in Microsoft Access, you start from a form and display it in Design View. To convert an existing form into a dialog box, display it in Design View first.

There are two types of dialog boxes: modal and modeless.

Practical Learning: Creating a Modal Dialog Box

1. To create a new form, on the (Form View) toolbar, click the arrow of the New Object button and click Insert -> Form



2. On the New Form dialog box, double-click Design View.

Lesson 9: Fundamental Windows Controls

- 3. Using your mouse, change the width of the form to **31**/2 and change its height to **17**/8
- 4. Save the form as **dlgAboutBCR**



Modal Dialog Boxes

A dialog box is characterized as modal if the user must close it before continuing with another task on the same application.

	6			B		
Times New F	Roman	• 12	▼ Wester	m	•	BZU
<u> </u>	· · · 1 ·	••••••	<u>2 · · · </u> ;	··· ș	· · ·] ·	•••4••
Because and beca a script, y language To let the sign, follo quotes. H <script h<br=""></script>						

The Date and Time

dialog box of WordPad is an example of a modal dialog box: if opened, the user must close it in order to continue using WordPad. In order to use a dialog box in your application, your first decision would be on why you want a dialog box. This means that you should analyze a scenario and define if the dialog box is necessary. Use a dialog box if you want the user to first terminate whatever task she would be performing. For example, if a user is performing a payment of an order processing, it is natural for the user to process and finish that payment before starting another task.

To create a modal dialog box in Microsoft Access, start a form in Design View and set its **Border Style** property value to **Dialog**. Unfortunately, as far as Windows is concerned, this only reduces the system buttons to the Close button only. A classic (or normal) dialog box would need neither a **Record Selectors** bar nor the record navigation buttons. Therefore, you should decide how the dialog box would be used. If you want a regular dialog box as those available on non-database applications, you should set the **Record Selectors**, the **Navigation Buttons** and the **Dividing Lines** properties to **No** each.

Practical Learning: Creating a Modal Dialog Box

- 1. On the Properties window, change the Caption property to **About the Bethesda Car Rental Application**
- 2. Change the **Record Selectors**, the **Navigation Buttons**, and the **Dividing Lines** properties to **No**
- 3. Change the Modal property to Yes
- 4. Change the Border Style property to Dialog
- 5. Save and preview the dialog box. When previewing the dialog box, on the main menu, click Window -> Size to Fit Form.
- 6. After viewing the dialog box, close it, using its Close button \blacksquare
- 7. Press F11 to display the Database Window. Click Forms to display the Forms section or the Forms property sheet
- 8. Right-click **AboutBCR** and click Rename.
- 9. Press Home, type **dlg** and press Enter. This changes the name of the dialog box to **dlgAboutBCR**.

Modeless Dialog Boxes

A dialog box is referred to as modeless if the user doesn't have to close it in order to continue using the application that owns the dialog box:

The **Find** dialog box of WordPad (also the **Find** dialog box of most applications) is an example of a modeless dialog box. If it is opened, the user doesn't have to close in order to use the application or the document in the background.

🚽 Document - WordPad 📃 🗖 🗙
File Edit View Insert Format Help
□☞日 @ 집 # ४ ☜ ඬ ∽ !!
Times New Roman 💽 12 💌 Western 💌 🖪 🖍 🖳 🕷
 O Symbol O Tahoma O Times New Roman T Times New Roman MT E O Trebuchet MS O Verdana T VisualUI O Webdings O Wingdings T Wingdings 2 A mathematical and the second second
±rr Wingdings 3
quotes. Fc <script browser="" find="" lat="" next<="" td="" what:=""></script>

Since the modeless dialog box doesn't display its button on the task bar, the user should know that the dialog box is opened. To make the presence of a modeless dialog box obvious to the user, it typically displays on top of its host application until the user closes it.

To create a modeless dialog box, or to convert a form into a modeless dialog box, when in Design View, set the **Popup** property to **Yes**. This makes sure that the user can work on another form and the modeless dialog box or form would remain on top of any other form of your database.

Practical Learning: Creating a Modal Dialog Box

- 1. To create a new form, in the Forms section of the Database Window, click New.
- 2. On the New Form dialog box, double-click Design View
- 3. Save the form as **dlgRentalRates**
- 4. On the Properties window, change the Caption to **Bethesda Car Rental -Rental Rates**
- 5. To make the form a modeless dialog box, set the **Record Selectors**, the **Navigation Buttons**, and the **Dividing Lines** properties to **No**
- 6. Set the **Pop Up** property to **Yes**
- 7. Set the Min Max Buttons property to None
- 8. Switch the dialog box to Form View.

- 9. Click any area of the Database Window. Notice that the modeless dialog box stays on top
- 10. Close the dialog box. When asked whether you want to save the form, click Yes

Controls: The Label

A label is a control used to display text on a form or dialog box. This control is for one of two main purposes: to provide quick information or to help the user identify another control on the form. In the strict sense, a label is usually not considered a Windows control. This is probably because the user has no other interaction with a label than to read the text the label displays. Therefore, the properties of a label are completely under your control: the user cannot change them (unless indirectly, that is under your control).

To create a label, on the Toolbox, click the **Label** button, click on the form or dialog box in Design View and type the desired text. After typing the text, you can control the properties of the label visually or using the Properties window.

Practical Learning: Creating a Modal Dialog Box

- 1. With the dlgAboutBCR form still open, to bring it up, on the main menu, click Window -> Bethesda Car Rental. Switch the form to Design View
- If the Toolbox is not displaying, on the main menu, click View -> Toolbox. On the Toolbox, click Label and click on the top-left section of the form (under the Detail bar)
- 3. Type Bethesda Car Rental (r)
- 4. On the Toolbox, click Label and click under the first label. Type Release 2.02
- 5. Add another label with the caption **This product is licensed to**
- 6. Add another label with the caption Bethesda Car Rental
- 7. Add another label with the caption **A Division of BCR Productions, Inc.**
- 8. Complete the dialog box and its labels as follows:



9. Save the form

Windows Controls: The Text Box

A text box is used to receive information from the user by her typing it into a rectangular box. A text box can also be used to simply display information to the user.

Although a text box is a good control by itself, it usually doesn't mean much to the user if she doesn't know what the box is used for. Therefore, you should always create a text box in conjunction with a label. The label is used to identify what the text box is used for. You should position the label on the left or on top of the text box. The rules of using the computer suggest that a text box be identified by the caption of its label. For example, if a label displays First Name, the text box on its right or under it would be called The First Name Text Box. If a label displays How Old Are You?, its corresponding text box would be called The How Old Are You Text Box. But, programmatically, the text box and its label are completely independent, in their behavior, functionality, and properties.

Practical Learning: Creating a Modal Dialog Box

- Heighten (make tall) the Form Footer section so it would have a height of 1" On the main menu, click View -> Form Header/Footer. On the Toolbox, click the Text Box button.
- 2. On the form, click under the Form Footer bar. Click the accompanying label to select it and press Delete.
- 3. Using the Properties window, move the text box the left and, using the Properties window, change its name to txtWarning
- 4. Change the Back Color to 14869218 (Red = Green = Blue = 226)

Lesson 9: Fundamental Windows Controls

5. Set the Special Effect to Shadowed and resize the text box as follows:



- 6. On the Form Design toolbar, click the Code button 2. On the Object combo box, select Form.
- 7. Implement the form's OnLoad event as follows:

```
Private Sub Form_Load()
Dim strWarning As String
strWarning = "Unauthorized reproduction or distribution " & ______"
of this program, any of its objects, code " & ______"
sections, or any of its portions, may result " & _____"
in severe civil and criminal penalties. Such " & _____"
activity will be prosecuted to the full extent " & _____"
possible of the applicable law."
txtWarning = strWarning
txtWarning.Enabled = False
txtWarning.Locked = True
End Sub
```

8. Save the form and preview it



9. Close the form and close Microsoft Access

Aesthetic Controls: The Rectangle

A rectangle is a pseudo-control used for one of two reasons. It can be used to create a group of items, thus serving as a host for other controls (actually, the rectangle in Microsoft Access is not a true host, not as compared to the panel available in Borland applications or Visual Basic/Visual C#; Microsoft Access rectangle is not a true host because it cannot "carry" the items inside of it).

To create or add a rectangle to a form or report, on the Toolbox, click Rectangle and click in the desired position on the form or report. One of the cutest features of the rectangle is based on its Special Effects

Practical Learning: Adding a Rectangle

1. Switch the form to Design View.

On the Toolbox, click the Rectangle button \square and click on the form

2. Using the rectangle handles (as we learned when designing controls), position the rectangle as follows:



3. Save and preview the form

Aesthetic Controls: The Line

A line in Microsoft Access is used to create a "physical" line, the same that would appear on a word processor or other type of application. Besides the line it draws, the Line has a few properties that can make it a valuable presence on a form or a report.

To create or add a line, on the Toolbox, click the Line button and click in the desired location on the form or report

Practical Learning: Adding a Line

- 1. Switch the form to Design View. On the Toolbox, click the Line button \searrow
- 2. On the form, draw a horizontal line just above the Unbound text box and under the Form Footer bar
- 3. Save and preview the form



4. Close the form

Aesthetic Controls: The Unbound Picture

Microsoft Access allows you to add a picture as an object not tied to a particular database field. Such a picture can be added to a form or report. The picture is referred to as unbound because it is not specifically related to a particular field of a table.

To add an unbound picture, on the Toolbox, click Unbound Object Frame and click on the form or report. A wizard would help you locate the picture and add it to the host object.

Practical Learning: Adding an Unbound Picture

- 1. Switch the form back to Design View and reduce the height of the header section to $\ensuremath{\mathbf{0}}$
- 2. On the Toolbox, click Unbound Object Frame
- 3. On the form, click on the left side under the Detail bar

Insert Object	? 🗙
C Create New	ОК
File: C:\My Documents\	Cancel
Browse E Link	🔲 Display as Icon
Result Inserts the contents of the file as an object into your document so that you may activate it using the application which created it.	

- 4. On the Insert Object dialog box, click the Create From File radio button and click the Browse... button
- 5. Locate and select bcrlogo1 and click Open
- 6. On the Insert Object dialog box, click OK
- 7. On the form, make sure the new picture is selected. On the Properties window set the Size Mode to Stretch
- 8. Set the Special Effect property to Flat and preview the form

🕄 About the Beth	esda Car Rental Application	×
	Bethesda Car Rental(r)	
	Release 2.02	
	This product is licensed to	
	Bethesda Car Rental	
	A Division of BCR Productions, Inc.	
	1204 Mendelson Rd Suite D6	
	Bethesda, MD 20854	
	Copyright (c) 2002 BCR Productions, Inc.	
Unauthorized repro program, any of its of its portions, may criminal penalties. S to the full extent po	duction or distribution of this objects, code sections, or any result in severe civil and Such activity will be prosecuted ssible of the applicable law.	-

9. Save and close the dialog box

Previous

Copyright © 2002-2004 FunctionX, Inc.

Next

Lesson 9: Fundamental Windows Controls

Relations-Based Controls

Introduction

One of the main and most valuable features of a relational database is the ability for objects to exchange information. This ensures that data is uniquely identified and not duplicated among the various objects of the application. Information entered on a form during data entry is made available to other forms and reports that would need it. This behavior and its related functionalities are configured and controlled through the relationships that various objects have towards each other.

VPractical Learning: Starting the Application

- 1. Start Microsoft Access and open the **Bethesda Car Rental3** database. Click Forms if necessary
- 2. Double-click the frmCars form to open it. Notice that we don't know who create this record althoug this information can be valuable during inventory
- 3. After viewing the form, close it.

The Fundamentals of Relational Database

Imagine you are a high school teacher. Every month you give homework to your students. After they have done their homework, you need to enter their grades in a database. If you have 40 to 200 students, it wouldn't make sense to create a new record for each student every time you need to enter his grades into the database. A relational database allow the student administrator to create students records, that is ,their personal information in a designated object (a form). When a teacher wants to enter grades for a particular student, he can open the appropriate form. This teacher should be able to select a select on the grade form instead of creating a new student. This also prevents duplicate information: if different teachers create a new record for each student, the student would end up with various records with possible different if not contradictory data.

To manage the flow of data from a table A to a table B, table B should have a field, like an ambassador, that represents the records of table A. Based on this, when somebody is processing any type of information on table B but needs information that is stored in table A, a control should allow the user to select, from table B, the information stored in table A without explicitly opening table A. To create such a scenario and relationship, the "ambassador" field of table B must have the same data type as the primary key of table A.

VPractical Learning: Starting the Application

- On the Database Window, click Tables and double-click tblCars to open the table in Datasheet View
- 2. After viewing the table, switch it to Design View.
- Right-click NbrOfDoors and click Insert Rows. Type CarCategoryID and press Tab
- 4. Set the Data Type as Number and press F6
- 5. In the lower section of the table, accept **Long Integer** as the **Field Size**. Change the **Caption** to **Category** and delete the 0 of Default Value

▦	tblCars : Table				X
	Field Nar	ne	Data Type	Description	
8	CarID		AutoNumber		
	CarNumber		Text		
	Make		Text		
	Model		Text		
	Year		Text		
	CarCategoryID		Number		
	NbrOfDoors		Number		
	AC		Yes/No		
1335	LOMEM		Vec/No		-
(F F C C C C C V V F F I I	General Lookup Field Size Format Decimal Places Input Mask Caption Default Value Validation Rule Validation Text Required Indexed	Long Inter Auto Category No Yes (Dupli	ger cates OK)		

- 6. Right-click CarNumber and click Build
- 7. In the Field Builder dialog box, in the Sample Tables, click Employees. In the Sample Fields, click EmployeeID and click OK
- 8. In the lower section of the table, change the Caption to Registered By
- 9. Right-click AC and click Insert Rows. Type TransmissionID
- 10. Change its Data Type to Number and make sure its Field Size is set to Long Integer. Set its Caption to Transmission and delete the 0 of the Default Value field
- 11. Save and close the table

Configuring Relationships

We have previously added fields to a table so the table can receive some of its data from other table. The only thing we did was to create fields, the table is still not aware of the other tables. We want to make sure that, if data is deleted on the table that holds the information, the dependent table must be made aware and dismiss the data that the originating table was providing. On the other hand, if data from the originating table changes, for example, if the name of an employee gets changed or updated in the Employees table, the change must be transparently updated on the new table.

VPractical Learning: Making a Form Active

- 1. On the main menu, click Tools -> Relationships...
- 2. Right-click in the Relationships window and click Show Table. In the Show Table dialog box, click tblEmployees and click Add
- 3. Double-click tblCars, tblCarCategories, and tblCarTransmissions
- 4. Click Close.

If you ever want to re-position a table, click and drag its title bar to the desired location.

5. On the tblEmployees table, click and drag EmployeeID. Drop it on top of EmployeeID in the tblCars table:



6. When the Relationships dialog box comes up, click the Enforce Referential Integrity check box and click the other two check boxes that would allow an update or a deletion of an employee from the tblEmployees table to be replicated in the tblCars table

elationships		? >
Table/Query:	Related Table/Query:	Create
tblEmployees	tblCars 🔺	
EmployeeID	EmployeeID	Cancel
		Join Type
Enforce Refere Cascade Upda Cascade Delete	ential Integrity te Related Fields e Related Records	
		97 STREET CONTRACTOR CONTRACTOR

- 7. Click Create.
- 8. From the tblCarCategories table, drag CarCategoryID and drop it on top of CarCategoryID in the tblCars table
- 9. On the Relationships dialog box, click all three check boxes and click Join Type
- 10. In the Join Properties dialog box, click the 2 radio button to ensure the database that one category from the tblCarCategories table can be applied to various cars in the tblCars table

Join Properties ? 🗙
C 1: Only include rows where the joined fields from both tables are equal.
 Include ALL records from 'tblCarCategories' and only those records from 'tblCars' where the joined fields are equal.
3: Include ALL records from 'tblCars' and only those records from 'tblCarCategories' where the joined fields are equal.
OK Cancel

- 11. Click OK and click Create.
- 12. Right-click the line that joins both EmployeeIDs and click Edit Relationship. Click Join Type and click the 2 radio button
- 13. Click OK twice
- 14. Drag TransmissionID from the tblCarTransmissions table to the TransmissionID of the tblCars table. Enforce both referential integrities and ensure that one transmission from the tblCarTransmissions table can be applied to valious records of the tblCars table



15. Close the Relationship window. When asked whether you want to save it, click Yes

List-Based Controls: The Combo Box

Besides the table that is at the core of a database, the operating system provides a few controls that are more adapted to display a list of items. One of these controls is the combo box.

A combo box is a window control that presents a list of items the user can select from. Visually, a combo box looks like a text box equipped with a down pointing arrow on its right side. By default, a combo box displays an item. In order to change the content of that box, different options are available. You can allow the user to type in the combo box and try to match what the user types with one of the items in the list. You can also make the user only select an item from the list by clicking the down pointing arrow. If the list is long and cannot display everything at the same time, a combo box is equipped with a vertical scroll bar.

There are two main types of combo boxes you will provide to your users. A classic combo box that is popular on regular applications displays a list of items. You create such a list and know its content all the time. Such a combo box is not bound to anything else. This is called an unbound combo box.

An unbound combo box can be created on a table, a form, or a report. To create an unbound combo box, if the table is displaying in Datasheet View, you can right-click a column and click Lookup Column... If the table is displaying in Design View, choose the Lookup Wizard as the Data Type for the desired field. In both cases a Lookup Wizard dialog box would come up and guide you

To create an unbound combo box for a form or report, open the object in Design View. On the Toolbox, click Combo Box and click on the form or report. If the Control Wizard button was selected in the Toolbox, in both cases a Combo Box Wizard would come up and guide you.

VPractical Learning: Creating an Unbound Combo Box

- 1. On the Database Window, click Tables. Right-click **tbITSDoubleShift** and click Design
- 2. Right-click **TuesdayIn** and click Insert-Rows. Type **MondayIn2** and press Tab.
- 3. For the Data Type, select Lookup Wizard...
- 4. In the first page of the Lookup Wizard, click the second radio button and click Next
- Click under Col1 and type 09:00 AM
- 6. Press the down arrow key and type 12:00 PM
- 7. Press the down arrow key and type 12:30 PM
- 8. Press Tab and type 05:00 PM
- 9. Click Next and click Finish
- 10. Save and close the table
- 11. On the Database Window, click Forms and double-click frmTSDoubleShift1 to open it:

ethesda Car Renta	l - Employees Tir	ne Sheet			
Time Sheet ID Employee	Jmber)	•	Date	[
	Time In	Time Out	Time In	Time Out	Total
Monday	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	0.00
Tuesday	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	0.00
Wednesday	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	0.00
Thursday	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	0.00
Friday	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	0.00
Saturday	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	0.00
Sunday	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	12:00 PM 💌	0.00
				Weekly Total	0.00
Notes					
ord: III I	, 3 > > >*	of 3			

- 12. Switch it to Design View. On the Form Design toolbar, click the Code button
- 13. The first thing we are going to do is to calculate the time an employee has worked for one day. Our time sheet is designed with two shifts for each day. There are two possible scenarios: an employee can work two shifts in one day or an employee can take a (too long) (lunch) break. We will divide the responsibility on the database developer and the user. As the developer, you will take care of checking and validating the time sequences. The user will have to make sure that, he did not take a break, he has to make sure that this reflects on the time sheet. If he puts a time a break between both shifts, the total time would subtract such an interception.

We will use a function that can receive the four time values of a date and calculate the total

time worked for the day. In the Code Editor, type the following function:

14. Once we can get the total for each day, we can retrieve the value displayed for each day and calculate the total time for the week. To get this total, type the following procedure:

```
Private Sub CalculateTotalTime()
    ' This function is used to calculate the total time
    ' based on the time displayed in the text box for each day
    Dim Monday As Double
    Dim Tuesday As Double
    Dim Wednesday As Double
    Dim Thursday As Double
    Dim Friday As Double
    Dim Saturday As Double
    Dim Sunday As Double
    Dim Total As Double
    ' Get the number of hours that is displaying
    ' in each text box of the corresponding day
    Monday = CDbl(txtMonday)
    Tuesday = CDbl(txtTuesday)
    Wednesday = CDbl(txtWednesday)
    Thursday = CDbl(txtThursday)
    Friday = CDbl(txtFriday)
    Saturday = CDbl(txtSaturday)
    Sunday = CDbl(txtSunday)
    ' Calculate the total number of hours for the week
    Total = Monday + Tuesday + Wednesday +
            Thursday + Friday + Saturday + Sunday
    ' Put the total in the Weekly Hours text box
    txtWeeklyTotal = Total
End Sub
```

15. Now we are ready to calculate to the actual time for each day. First, we must validate the time values that the user selects on a combo box, depending on the combo box. Consider a day like Monday. The starting time that the user selects should not be higher than the end time. This means that a user cannot state that he worked from 12PM to 9AM in one day. Logically, such a sequence doesn't make sense. Therefore, the first time in for Monday must not occur after the first time out for Monday.

To implement this algorithm, in the Object combo box, select **MondayIn1** and, in the Procedure combo box, select **AfterUpdate**.

16. Implement the **AfterUpdate** event as follows:

```
Private Sub MondayIn1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    ' Get the time combo boxes that are involved on this AfterUpdate
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
    ' If the user has AfterUpdated the time on MondayIn1,
    ' make sure the new time doesn't occur after MondayOut1
    ' if the user tries this, set MondayIn1 to the value of MondayOut1
    If OutSel1 < InSel1 Then
       MondayIn1 = MondayOut1
    End If
    ' Get the current selections in the combo boxes for this day
    ' Convert their values to date
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
    InSel2 = CDate(MondayIn2)
    OutSel2 = CDate(MondayOut2)
    ' Send the time values the CalcTodayTime function for processing
    txtMonday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    ' Call the CalculateTotalTime procedure to display the total time
    CalculateTotalTime
End Sub
```

- 17. When it comes to the first time out value, we need to fulfill two conditions: a)the value of the first time out must not be lower than that of the first time in; b) the value of the first time out must not be higher than that of the second time in. To implement this behavior, on the Object combo box, select **MondayOut1** and, in the Procedure combo box, select **AfterUpdate**.
- 18. Implement the **AfterUpdate** event as follows:

```
Private Sub MondayOut1_AfterUpdate()
   Dim InSel1 As Date
   Dim OutSel1 As Date
   Dim InSel2 As Date
   Dim OutSel2 As Date
   InSel1 = CDate(MondayIn1)
   OutSel1 = CDate(MondayOut1)
   InSel2 = CDate(MondayIn2)
   ' If the time selected in MondayOut1 occurs before MondayIn1,
   ' set it to the time of MondayIn1
   If OutSel1 < InSel1 Then
        MondayOut1 = MondayIn1 ' Format(InSel1, "hh:nn AM/PM")</pre>
```

```
End If
    ' If the selected time occurs after MondayIn2,
    ' set it to the time in MondayIn2
    If OutSel1 > InSel2 Then
        MondayOut1 = MondayIn2
    End If
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
    InSel2 = CDate(MondayIn2)
    OutSel2 = CDate(MondayOut2)
    txtMonday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
```

19. These two algorithms will be applied to all the other combo boxes. Therefore, implement their **OnChange** events as follows:

```
Option Compare Database
Option Explicit
Private Function CalcTodayTime(TimeIn1 As Date, TimeOut1 As Date,
                               TimeIn2 As Date, TimeOut2 As Date) As Double
    ' This variable will hold the time difference for the first shift
    Dim dblShift1 As Double
    ' This variable will hold the time difference for the second shift
    Dim dblShift2 As Double
    dblShift1 = DateDiff("n", TimeIn1, TimeOut1) / 60
    dblShift2 = DateDiff("n", TimeIn2, TimeOut2) / 60
    ' The time worked is the total of both shifts
    CalcTodayTime = dblShift1 + dblShift2
End Function
Private Sub CalculateTotalTime()
    ' This function is used to calculate the total time
    ' based on the time displayed in the text box for each day
    Dim Monday As Double
    Dim Tuesday As Double
    Dim Wednesday As Double
    Dim Thursday As Double
    Dim Friday As Double
    Dim Saturday As Double
    Dim Sunday As Double
    Dim Total As Double
    ' Get the number of hours that is displaying
    ' in each text box of the corresponding day
    Monday = CDbl(txtMonday)
    Tuesday = CDbl(txtTuesday)
    Wednesday = CDbl(txtWednesday)
    Thursday = CDbl(txtThursday)
    Friday = CDbl(txtFriday)
    Saturday = CDbl(txtSaturday)
```

Sunday = CDbl(txtSunday)

```
' Calculate the total number of hours for the week
    Total = Monday + Tuesday + Wednesday +
            Thursday + Friday + Saturday + Sunday
    ' Put the total in the Weekly Hours text box
    txtWeeklyTotal = Total
End Sub
Private Sub MondayIn1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    ' Get the time combo boxes that are involved on this AfterUpdate
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
    ' If the user has AfterUpdated the time on MondayIn1,
    ' make sure the new time doesn't occur after MondayOut1
    ' if the user tries this, set MondayIn1 to the value of MondayOut1
    If OutSell < InSell Then
       MondayIn1 = MondayOut1
    End If
    ' Get the current selections in the combo boxes for this day
    ' Convert their values to date
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
    InSel2 = CDate(MondayIn2)
    OutSel2 = CDate(MondayOut2)
    ' Send the time values the CalcTodayTime function for processing
    txtMonday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    ' Call the CalculateTotalTime procedure to display the total time
    CalculateTotalTime
End Sub
Private Sub MondayIn2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    OutSel1 = CDate(MondayOut1)
    InSel2 = CDate(MondayIn2)
    OutSel2 = CDate(MondayOut2)
    If InSel2 < OutSel1 Then
       MondayIn2 = MondayOut1
    End If
    If InSel2 > OutSel2 Then
        MondayIn2 = MondayOut2
    End If
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
```

```
InSel2 = CDate(MondayIn2)
    OutSel2 = CDate(MondayOut2)
    txtMonday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub MondayOut1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
    InSel2 = CDate(MondayIn2)
    ' If the time selected in MondayOut1 occurs before MondayIn1,
    ' set it to the time of MondayIn1
    If OutSell < InSel1 Then
        MondayOut1 = MondayIn1 ' Format(InSel1, "hh:nn AM/PM")
    End If
    ' If the selected time occurs after MondayIn2,
    ' set it to the time in MondayIn2
    If OutSel1 > InSel2 Then
        MondayOut1 = MondayIn2
    End If
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
    InSel2 = CDate(MondayIn2)
    OutSel2 = CDate(MondayOut2)
    txtMonday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub MondayOut2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel2 = CDate(MondayIn2)
    OutSel2 = CDate(MondayOut2)
    If OutSel2 < InSel2 Then
        MondayOut2 = MondayIn2
    End If
    InSel1 = CDate(MondayIn1)
    OutSel1 = CDate(MondayOut1)
    InSel2 = CDate(MondayIn2)
    OutSel2 = CDate(MondayOut2)
    txtMonday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
```

```
Private Sub TuesdayIn1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(TuesdayIn1)
    OutSel1 = CDate(TuesdayOut1)
    If OutSell < InSel1 Then
        TuesdayIn1 = TuesdayOut1
    End If
    InSel1 = CDate(TuesdayIn1)
    OutSel1 = CDate(TuesdayOut1)
    InSel2 = CDate(TuesdayIn2)
    OutSel2 = CDate(TuesdayOut2)
    txtTuesday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub TuesdayIn2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    OutSel1 = CDate(TuesdayOut1)
    InSel2 = CDate(TuesdayIn2)
    OutSel2 = CDate(TuesdayOut2)
    If InSel2 < OutSel1 Then
        TuesdayIn2 = TuesdayOut1
    End If
    If InSel2 > OutSel2 Then
        TuesdayIn2 = TuesdayOut2
    End If
    InSel1 = CDate(TuesdayIn1)
    OutSel1 = CDate(TuesdayOut1)
    InSel2 = CDate(TuesdayIn2)
    OutSel2 = CDate(TuesdayOut2)
    txtTuesday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub TuesdayOut1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(TuesdayIn1)
    OutSel1 = CDate(TuesdayOut1)
    InSel2 = CDate(TuesdayIn2)
```

```
If OutSell < InSell Then
        TuesdayOut1 = TuesdayIn1
    End If
    If OutSel1 > InSel2 Then
        TuesdayOut1 = TuesdayIn2
    End If
    InSel1 = CDate(TuesdayIn1)
    OutSel1 = CDate(TuesdayOut1)
    InSel2 = CDate(TuesdayIn2)
    OutSel2 = CDate(TuesdayOut2)
    txtTuesday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub TuesdayOut2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel2 = CDate(TuesdayIn2)
    OutSel2 = CDate(TuesdayOut2)
    If OutSel2 < InSel2 Then
        TuesdayOut2 = TuesdayIn2
    End If
    InSel1 = CDate(TuesdayIn1)
    OutSel1 = CDate(TuesdayOut1)
    InSel2 = CDate(TuesdayIn2)
    OutSel2 = CDate(TuesdayOut2)
    txtTuesday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub WednesdayIn1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(WednesdayIn1)
    OutSel1 = CDate(WednesdayOut1)
    If OutSel1 < InSel1 Then
        WednesdayIn1 = WednesdayOut1
    End If
    InSel1 = CDate(WednesdayIn1)
    OutSel1 = CDate(WednesdayOut1)
    InSel2 = CDate(WednesdayIn2)
    OutSel2 = CDate(WednesdayOut2)
    txtWednesday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
```

End Sub

```
Private Sub WednesdayIn2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    OutSel1 = CDate(WednesdayOut1)
    InSel2 = CDate(WednesdayIn2)
    OutSel2 = CDate(WednesdayOut2)
    If InSel2 < OutSel1 Then
        WednesdayIn2 = WednesdayOut1
    End If
    If InSel2 > OutSel2 Then
        WednesdayIn2 = WednesdayOut2
    End If
    InSel1 = CDate(WednesdayIn1)
    OutSel1 = CDate(WednesdayOut1)
    InSel2 = CDate(WednesdayIn2)
    OutSel2 = CDate(WednesdayOut2)
    txtWednesday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub WednesdayOut1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(WednesdayIn1)
    OutSel1 = CDate(WednesdayOut1)
    InSel2 = CDate(WednesdayIn2)
    If OutSel1 < InSel1 Then
        WednesdayOut1 = WednesdayIn1
    End If
    If OutSel1 > InSel2 Then
        WednesdayOut1 = WednesdayIn2
    End If
    InSel1 = CDate(WednesdayIn1)
    OutSel1 = CDate(WednesdayOut1)
    InSel2 = CDate(WednesdayIn2)
    OutSel2 = CDate(WednesdayOut2)
    txtWednesday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub WednesdayOut2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
```

```
Dim OutSel2 As Date
    InSel2 = CDate(WednesdayIn2)
    OutSel2 = CDate(WednesdayOut2)
    If OutSel2 < InSel2 Then
        WednesdayOut2 = WednesdayIn2
    End If
    InSel1 = CDate(WednesdayIn1)
    OutSel1 = CDate(WednesdayOut1)
    InSel2 = CDate(WednesdayIn2)
    OutSel2 = CDate(WednesdayOut2)
    txtWednesday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub ThursdayIn1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(ThursdayIn1)
    OutSel1 = CDate(ThursdayOut1)
    If OutSel1 < InSel1 Then
        ThursdayIn1 = ThursdayOut1
    End If
    InSel1 = CDate(ThursdayIn1)
    OutSel1 = CDate(ThursdayOut1)
    InSel2 = CDate(ThursdayIn2)
    OutSel2 = CDate(ThursdayOut2)
    txtThursday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub ThursdayIn2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    OutSel1 = CDate(ThursdayOut1)
    InSel2 = CDate(ThursdayIn2)
    OutSel2 = CDate(ThursdayOut2)
    If InSel2 < OutSel1 Then
        ThursdayIn2 = ThursdayOut1
    End If
    If InSel2 > OutSel2 Then
        ThursdayIn2 = ThursdayOut2
    End If
    InSel1 = CDate(ThursdayIn1)
    OutSel1 = CDate(ThursdayOut1)
```

InSel2 = CDate(ThursdayIn2)

```
OutSel2 = CDate(ThursdayOut2)
    txtThursday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub ThursdayOut1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(ThursdayIn1)
    OutSel1 = CDate(ThursdayOut1)
    InSel2 = CDate(ThursdayIn2)
    If OutSell < InSell Then
        ThursdayOut1 = ThursdayIn1
    End If
    If OutSel1 > InSel2 Then
        ThursdayOut1 = ThursdayIn2
    End If
    InSel1 = CDate(ThursdayIn1)
    OutSel1 = CDate(ThursdayOut1)
    InSel2 = CDate(ThursdayIn2)
    OutSel2 = CDate(ThursdayOut2)
    txtThursday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub ThursdayOut2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel2 = CDate(ThursdayIn2)
    OutSel2 = CDate(ThursdayOut2)
    If OutSel2 < InSel2 Then
        ThursdayOut2 = ThursdayIn2
    End If
    InSel1 = CDate(ThursdayIn1)
    OutSel1 = CDate(ThursdayOut1)
    InSel2 = CDate(ThursdayIn2)
    OutSel2 = CDate(ThursdayOut2)
    txtThursday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub FridayIn1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
```

```
Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(FridayIn1)
    OutSel1 = CDate(FridayOut1)
    If OutSel1 < InSel1 Then
        FridayIn1 = FridayOut1
    End If
    InSel1 = CDate(FridayIn1)
    OutSel1 = CDate(FridayOut1)
    InSel2 = CDate(FridayIn2)
    OutSel2 = CDate(FridayOut2)
    txtFriday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub FridayIn2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    OutSel1 = CDate(FridayOut1)
    InSel2 = CDate(FridayIn2)
    OutSel2 = CDate(FridayOut2)
    If InSel2 < OutSel1 Then
       FridayIn2 = FridayOut1
    End If
    If InSel2 > OutSel2 Then
        FridayIn2 = FridayOut2
    End If
    InSel1 = CDate(FridayIn1)
    OutSel1 = CDate(FridayOut1)
    InSel2 = CDate(FridayIn2)
    OutSel2 = CDate(FridayOut2)
    txtFriday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub FridayOut1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(FridayIn1)
    OutSel1 = CDate(FridayOut1)
    InSel2 = CDate(FridayIn2)
    If OutSell < InSel1 Then
        FridayOut1 = FridayIn1
    End If
```

If OutSel1 > InSel2 Then

```
FridayOut1 = FridayIn2
    End If
    InSel1 = CDate(FridayIn1)
    OutSel1 = CDate(FridayOut1)
    InSel2 = CDate(FridayIn2)
    OutSel2 = CDate(FridayOut2)
    txtFriday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub FridayOut2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel2 = CDate(FridayIn2)
    OutSel2 = CDate(FridayOut2)
    If OutSel2 < InSel2 Then
        FridayOut2 = FridayIn2
    End If
    InSel1 = CDate(FridayIn1)
    OutSel1 = CDate(FridayOut1)
    InSel2 = CDate(FridayIn2)
    OutSel2 = CDate(FridayOut2)
    txtFriday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub SaturdayIn1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(SaturdayIn1)
    OutSel1 = CDate(SaturdayOut1)
    If OutSell < InSel1 Then
        SaturdayIn1 = SaturdayOut1
    End If
    InSel1 = CDate(SaturdayIn1)
    OutSel1 = CDate(SaturdayOut1)
    InSel2 = CDate(SaturdayIn2)
    OutSel2 = CDate(SaturdayOut2)
    txtSaturday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub SaturdayIn2 AfterUpdate()
    Dim InSell As Date
```

```
Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    OutSel1 = CDate(SaturdayOut1)
    InSel2 = CDate(SaturdayIn2)
    OutSel2 = CDate(SaturdayOut2)
    If InSel2 < OutSel1 Then
        SaturdayIn2 = SaturdayOut1
    End If
    If InSel2 > OutSel2 Then
        SaturdayIn2 = SaturdayOut2
    End If
    InSel1 = CDate(SaturdayIn1)
    OutSel1 = CDate(SaturdayOut1)
    InSel2 = CDate(SaturdayIn2)
    OutSel2 = CDate(SaturdayOut2)
    txtSaturday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub SaturdayOut1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(SaturdayIn1)
    OutSel1 = CDate(SaturdayOut1)
    InSel2 = CDate(SaturdayIn2)
    If OutSell < InSel1 Then
        SaturdayOut1 = SaturdayIn1
    End If
    If OutSel1 > InSel2 Then
        SaturdayOut1 = SaturdayIn2
    End If
    InSel1 = CDate(SaturdayIn1)
    OutSel1 = CDate(SaturdayOut1)
    InSel2 = CDate(SaturdayIn2)
    OutSel2 = CDate(SaturdayOut2)
    txtSaturday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub SaturdayOut2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel2 = CDate(SaturdayIn2)
    OutSel2 = CDate(SaturdayOut2)
```

```
If OutSel2 < InSel2 Then
        SaturdayOut2 = SaturdayIn2
    End If
    InSel1 = CDate(SaturdayIn1)
    OutSel1 = CDate(SaturdayOut1)
    InSel2 = CDate(SaturdayIn2)
    OutSel2 = CDate(SaturdayOut2)
    txtSaturday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub SundayIn1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(SundayIn1)
    OutSel1 = CDate(SundayOut1)
    If OutSell < InSel1 Then
        SundayIn1 = SundayOut1
    End If
    InSel1 = CDate(SundayIn1)
    OutSel1 = CDate(SundayOut1)
    InSel2 = CDate(SundayIn2)
    OutSel2 = CDate(SundayOut2)
    txtSunday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub SundayIn2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    OutSel1 = CDate(SundayOut1)
    InSel2 = CDate(SundayIn2)
    OutSel2 = CDate(SundayOut2)
    If InSel2 < OutSel1 Then
        SundayIn2 = SundayOut1
    End If
    If InSel2 > OutSel2 Then
        SundayIn2 = SundayOut2
    End If
    InSel1 = CDate(SundayIn1)
    OutSel1 = CDate(SundayOut1)
    InSel2 = CDate(SundayIn2)
    OutSel2 = CDate(SundayOut2)
    txtSunday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
```

```
CalculateTotalTime
End Sub
Private Sub SundayOut1 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel1 = CDate(SundayIn1)
    OutSel1 = CDate(SundayOut1)
    InSel2 = CDate(SundayIn2)
    If OutSell < InSel1 Then
        SundayOut1 = SundayIn1
    End If
    If OutSel1 > InSel2 Then
        SundayOut1 = SundayIn2
    End If
    InSel1 = CDate(SundayIn1)
    OutSel1 = CDate(SundayOut1)
    InSel2 = CDate(SundayIn2)
    OutSel2 = CDate(SundayOut2)
    txtSunday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
Private Sub SundayOut2 AfterUpdate()
    Dim InSell As Date
    Dim OutSell As Date
    Dim InSel2 As Date
    Dim OutSel2 As Date
    InSel2 = CDate(SundayIn2)
    OutSel2 = CDate(SundayOut2)
    If OutSel2 < InSel2 Then
        SundayOut2 = SundayIn2
    End If
    InSel1 = CDate(SundayIn1)
    OutSel1 = CDate(SundayOut1)
    InSel2 = CDate(SundayIn2)
    OutSel2 = CDate(SundayOut2)
    txtSunday = CalcTodayTime(InSel1, OutSel1, InSel2, OutSel2)
    CalculateTotalTime
End Sub
```

20. Save, test, and close the form

The Bound Combo Box

A bound combo box is one whose list is linked to an external object. This type of combo box allows a user to select data that is stored on a table other than the table the user is processing. The ease of creating and configuring this control is one of the strengths of Microsoft Access.

To create a bound combo box on a table, whether it is opened in Datasheet View or in Design View, follow the first step as for the unbound combo box. On the first page of the Lookup Wizard, select the first radio button.

To create a bound combo box on a form or report, while the object is opened in Design View, on the Toolbox, click the the Combo Box button and click on the form or report. If the Control Wizard button was selected on the Toolbox, when the Combo Box Wizard comes up, select the first radio button and proceed.

VPractical Learning: Creating an Unbound Combo Box

- 1. On the Database Window, click Forms.
- 2. Double-click frmCars to open it. After viewing the form, switch it to Form View.
- 3. On the Toolbox, make sure the Control Wizards button is down. Click Combo Box
- 4. On the form, click just under the Year text Box

88	frmCars : Form			
		• 1 • • • • • • • 2		3 • • • 1 • • • 4
	Form Header			
ŀ	Beth	esda Car I	Rental - (Cars Cata
	🗲 Detail			
$\left[- \right]$			******	******
1:	Car ID:	CarlD		
	Car #	CarNumber	Available	
E	Make	Make	Model	Model
1.	Year	Year	Nbr of Doors	NbrOfDoors
:				
-	Car Option	15		•
				► //

- 5. On the first page of the Combo Box Wizard, make sure the first radio button is selected and click Next
- 6. In the second page, click tblCarTransmissions and click Next.
- 7. On the 3rd page, click the Select All button 🔛 and click Next twice
- 8. On the 5th page, click the arrow of the combo box and select TransmissionID

XXXXXXXX XXXX XXXX XXXX XXXX XXXX XXXX	Microsoft Access can store the selected value from your combo box in your database, or remember the value so you can use it later to perform a task. Remember the value for later use. Store that value in this field:
	Cancel < <u>B</u> ack <u>Next</u> > Finish

- 9. Click Next and click Finish
- 10. While the combo box itself is still selected, on the Properties window, change its name to **cboTransmissions**
- 11. Change the caption of its label to Transmission and set its font color to blue
- 12. In the same way, create a combo box for the car category and position it under the NbrOfDoors text box. Name it **cboCategories**
- 13. To create a navigation combo boxSave and close the form

List-Based Controls: List Boxes

A list box is a control that presents a list of items to the user who can make his selection by clicking on the list. Depending on how the list is configured, the user can click one item at a time or select many items from the list. Like a combo box, a list box is a control whose main purpose is to display a list of items. Unlike a combo box, a list box always displays its items. If the number items is higher than the height of the control can display, the list box gets equipped with a (vertical) scroll bar (the operating system takes care of displaying a scroll bar when necessary.

On a database (such as Microsoft Access), you cannot create a list box on a table. To create a list box on a form or report, display the object in Design View. On the Toolbox, click List Box and click in the desired section on the form or report. If the Controls Wizard was selected on the Toolbox, this would start the List Box Wizard that can help you create an configure your list box. Like a combo box, there are two types of list boxes you can create in your database: bound or unbound. A bound list box is one whose data originates from another object such as a table or a query. A list box is considered unbound when its data is not tied to any external object, which means the list box controls its own list.

Probably the most important feature of a list box is the list of items that composes it. If you are using the List Box Wizard to create your list box, you can select the first radio button of the List Box Wizard to create a bound list or the second radio button to create an unbound list of items.

VPractical Learning:

- 1. On the Forms section or property sheet of the Database Window, double-click dlgRentRates to open it.
- 2. After viewing it, to switch it to Design View, right-click its title bar and click Form Design
- 3. On the Toolbox, make sure the Controls Wizard is selected . Click the List Box button .
- 4. Click on the top left section of the form
- 5. On the first page of the List Box Wizard, make sure the first radio button is selected and click Next

List Box Wizard	Which table or query should provide the values for your list box?
	tblCarCategories tblCarTransmissions tblCars tblCarsCondition tblCustomers tblEmployees tblPaymentMethods tblPayments tblProcessOrder
	View Tables C Queries C Both
	Cancel < <u>B</u> ack <u>N</u> ext > <u>F</u> inish

- 6. In the list of tables, click **tblCarCategories** and click Next
- 7. On the 3rd page, click the Select All button
| | Which fields contain the values you want included in your list box?
The fields you select become columns in your list box. |
|-------------------|---|
| Available Fields: | Selected Fields:
CarCategoryID
Category
DailyRate
WeeklyRate
MonthlyRate
WeekendRate |
| | Cancel < <u>B</u> ack <u>N</u> ext > <u>Finish</u> |

8. Click Next

9. On the 4th page, double-click the lines between each pair of columns

List Ho To rigi	Box Wizard w wide would you adjust the width ht edge of the col Hide key column	u like the column: of a column, drag umn heading to g ı (recommended)	s in your list box? j its right edge to jet the best fit.	the width you war	it, or double-click the	
	Category	DailyRate	WeeklyRate	MonthlyRate	WeekendRate	
	Economy	\$32.95	\$29.75	\$22.95	\$19.95	
	Compact	\$39.95	\$34.75	\$24.95	\$29.95	
	Standard	\$45.95	\$39.75	\$35.95	\$34.95	
	Full Size	\$49.95	\$42.75	\$35.95	\$38.95	
	Mini Van	\$55.95	\$50.95	\$45.95	\$42.95	
	SUV	\$55.95	\$50.95	\$45.95	\$42.95	-I
	1+ -	i to on	tor an	100.05	100.05	Ľ
		(Ca	ncel) <	<u>B</u> ack <u>N</u> ex	t> <u>F</u> inish	

10. Click Next

List Box Wizard	
	What label would you like for your list box? Car Rental Rates
	Those are all the answers the wizard needs to create your list box.
	Display Help on customizing the list box.
	Cancel < <u>B</u> ack <u>M</u> ext> <u>F</u> inish

- 11. Change the label to Car Rental Rates and press Enter
- 12. Change the label's font style to Bold and arrange the form as follows:

8 d	lg	R	en	ta	all	Ra	at	e	5	: 1	Fo	r	n																										1	×
			•	•	I			•	•		1	•		 •	T		•	•	2		•	•	1	•		•	•	3	•	•		•	•	•	4	•	•	•	1	•
	4	FC	e	ai			3		ŝ	8																													_	
:		C	at	:: :F	le	n	ta	iii dii	B	j	te	\$		 																										
-		U	nb	0	In	d																									 									
:																																								
																																						100000		
-																																								
				•••					 [•••	ŀ			 		 			•	 					•••			<u>. .</u>			 	 			· [·					-

13. Using the Properties window for the list box, set its Column Heads property to Yes

Car Rental	Rates				
Category	Daily	Weekly	Monthly	Weekend	
Economy	\$32.95	\$29.75	\$22.95	\$19.95	
Compact	\$39.95	\$34.75	\$24.95	\$29.95	
Standard	\$45.95	\$39.75	\$35.95	\$34.95	
Full Size	\$49.95	\$42.75	\$35.95	\$38.95	
Mini Van	\$55.95	\$50.95	\$45.95	\$42.95	
SUV	\$55.95	\$50.95	\$45.95	\$42.95	
Truck	\$42.95	\$35.75	\$30.95	\$30.95	
Van	\$69.95	\$59.75	\$50.75	\$49.95	

14. Save, preview, and close the form

Previous Copyright © 2002-2004 FunctionX, Inc. Next



Introduction

In the world of Microsoft Windows operating systems, a button is an object that a user clicks either in response to a question or to initiate an action. In Microsoft Access, there are three main types of buttons: the option button, the check box, and the command button.

Window Controls: The Radio Buttons

Also referred to as a radio button, an option button is a Windows control made of two sections: a round box O and a label. The label informs the user as to what the control is used for. The user makes her decision by selecting or clicking the round box \odot . In practical usage, a radio button is accompanied by one or more other radio buttons that appear and behave as a group. The user decides which button is valid by selecting only one of them. When the user clicks one button, its round box fills with a (big) dot: \odot . When one button in the group is selected, the other round buttons of the (same) group are emptied. The user can select another button by clicking a different choice, which empties the previous selection. This technique of selecting is referred to as mutually-exclusive.

Because they behave as a group, radio buttons are created using another object that can "hold" or host them. In Microsoft Access, this is usually accomplished by the presence of a Group Box object.

Practical Learning: Creating Option Buttons

- 1. Start Microsoft Access and create a Blank Database named Computer Store
- On the main menu, click Insert -> Table. On the New Table dialog box, doubleclick Design View
- 3. Type the first field as OrderProcessingID
- 4. Change its Data Type to AutoNumber and make it the Primary Key of the table
- 5. Set the name of the second field as Monitor and press Tab
- 6. For its Data Type, select LookupWizard...
- 7. On the first page of the Lookup Wizard, click the second radio button and click Next
- 8. Click the empty field under Col1 and Type Monitor1
- 9. Press Tab, type Monitor2 and complete the list as follows:

Lookup Wizard
What values do you want to see in your lookup column? Enter the number of columns you want in the list, and then type the values you want in each cell. To adjust the width of a column, drag its right edge to the width you want, or double-click the right edge of the column heading to get the best fit.
Number of columns: 1
Col1 Monitor1 Monitor2 Monitor3 Monitor4 *
Cancel < <u>B</u> ack <u>N</u> ext > <u>F</u> inish

- 10. Click Next
- 11. Accept the label as Monitor and click Finish
- 12. Close the table. When asked whether you want to save it, click Yes.
- 13. Type **tblOrders** and press Enter.
- 14. On the main menu, click Insert -> Form. On the New Form dialog box, click Design View. On the table or query combo box, select tblOrders and click OK
- 15. On the Toolbox, click Text Box and click on the top left section of the form.
- 16. Change the label to Order ID and, on the Properties window, change the Control Source of the text box to Monitor
- 17. On the Toolbox, make sure the Control Wizards button is down and click Option Group
- 18. Click on the form.
- 19. When the first page of the Option Group Wizard displays, click the empty box under Label Names and type 15 Inch Color
- 20. Press Tab and complete the list as follows:

• XXX XXX	An option group contains a set of option buttons, check boxes, or toggle buttons. You can choose only one option					
	What label do you want for each option?					
	Label Names:					
	15 Inch Color					
	17 Inch Color					
	19 Inch Color					
	15 Inch Flat Panel					
	17 Inch Flat Panel					

- 21. Click Next.
- 22. On the second page of the wizard, leave the first option as default and click Next.
- 23. On the third page, accept the suggested structure of the list and click Next.
- 24. On the fourth page, click the arrow of the combo box and select Monitor

• ***	You can either store the value of a selected option in a field, or use the value later to perform a task such as printing a report.
C xxx xxx C xxxx xxxx	What do you want to do with the value of a selected option?
	C Save the value for later use.
××× ××× <mark>×××</mark> ××× ××× ××× <mark>×××</mark> ×××	Store the value in this field: Monitor

- 25. Click Next
- 26. On the fifth page, accept the suggested layout and click Next
- 27. On the sixth page, change the label to Monitor and click Finish
- 28. Save the form as frmOrders and preview it



29. Close the form

Windows Controls: Check Boxes

A check box is a control that allows the user to validate or invalidate an option. Although it can appear by itself, a check box sometimes comes in a group with others, allowing the user to select as many choices as are available. Depending on the compiler or the environment, a little square appears with a check box \boxtimes and a label. The user makes her selection by clicking in the square which toggles a check mark \square . Toggling means if the square were empty \square , after clicking it, a check mark would appear in it \boxtimes ; otherwise, the check mark would be removed.

From the user's standpoint, a check box is selected when its check mark is set; and the item is not selected when its square is empty.

From the developer standpoint, a check mark has two (Boolean) values: true or false (or TRUE or FALSE, or True or False). When a check mark is selected, its value is true; otherwise, its value is false. Those are the primary concerns you will have with check marks.

A check mark is a special button and can be programmed as a regular button. Therefore, depending on your needs, you can display or hide it at will; you can enable and disable it as necessary; you can adjust its behavior depending on other controls on the same form or the same application.

Check boxes provide a non-exclusive choice, which means each can behave as independent as you want with regard to the other check boxes of the same group. Therefore, although not recommended, you can create check boxes anywhere on the form or report. If you plan to use a group of check boxes, it is recommended that you include them in rectangular container so their belonging to the same group will be obvious to the user. The group can be hosted by a **GroupBox** (recommended) or a rectangle.

To create a check box in Microsoft Access, if the check box will be tied to a value on a table, set the Data Type of the field to Yes/No.

Practical Learning: Creating Check Boxes

- 1. Open the tblOrders table in Design View
- 2. Click the first empty field under Monitor and type HasPowerCord
- 3. Set its Data Type to Yes
- 4. Save and close the table
- 5. Open the **frmOrders** in Design View
- Display the Field List window.
 Drag HasPowerCord from the Field List and drop it on the form.
- 7. Change its label to **Has Power Cord?** and position it as follows:

🗉 frmOrders : Form
• • • • • • • • • • • • • • • • •
Order ID: OrderProcessingID Monitor 15 Inch:Color 1 • 15 Inch:Color 1 • 17 Inch:Color • 19 Inch:Color • 13 Inch:Color • 17 Inch:Flat:Panel • 17 Inch:Flat:Panel • 17 Inch:Flat:Panel • 17 Inch:Flat:Panel • Has Power:Cord ? •

- 8. Save and close the form
- 9. Open the Ocean Service Computer Store database and open the frmOrderProcessing form

Primary Configuration Accessories Operating System • Red Hat Linux 8.0 • Caldera OpenLinux Workstation • SuSE Linux 8.1 • MS Windows XP Home Edition • MS Windows 2000 Professional Hardware/Software Support AntiVirus • Compressor/Decompressor	Ating System Keyboard Standard JD Ergonomic Wireless Multi-Random Speakers Surge Protector Mouse Regular PS/2 2-Button Ergo Track-Ball Wireless Mouse Pad	Notes:	
Total Order:		_]	

- 10. After viewing the form, switch it to Design View
- 11. On the Form Design toolbar, click the Code button
- 12. Instead of performing a calculation for each control on the form, we will use a central procedure that can take care of all calculations. Therefore, in the Code Editor, type the following procedure

```
Public Sub CalculateOrder()
   Dim curProcessor As Currency
   Dim curHardDrive As Currency
   Dim curRAM As Currency
   Dim curRemovable As Currency
   Dim curMonitor As Currency
   Dim curNIC As Currency
    Dim curPrinter As Currency
   Dim curFax As Currency
    Dim curCopier As Currency
   Dim curScanner As Currency
    Dim curOS As Currency
    Dim curKeyboard As Currency
    Dim curSpeakers As Currency
   Dim curSurge As Currency
    Dim curAntiVirus As Currency
   Dim curCompressor As Currency
    Dim curZIPDrive As Currency
   Dim curModem As Currency
    Dim curMouse As Currency
    Dim curMousePad As Currency
    Dim TotalOrder As Currency
```

```
Select Case Processors.Value
   Case 1
        curProcessor = 128
    Case 2
        curProcessor = 150
    Case 3
        curProcessor = 65
    Case 4
        curProcessor = 175
    Case 5
       curProcessor = 185
End Select
Select Case HardDrives.Value
   Case 1
       curHardDrive = 65
    Case 2
       curHardDrive = 105
    Case 3
       curHardDrive = 75
    Case 4
       curHardDrive = 80
    Case 5
       curHardDrive = 118
End Select
Select Case RAM.Value
    Case 1
       curRAM = 30
    Case 2
       curRAM = 56
    Case 3
       curRAM = 115
    Case 4
       curRAM = 312
End Select
Select Case RemovableMedia.Value
   Case 1
       curRemovable = 18
    Case 2
        curRemovable = 130
    Case 3
       curRemovable = 45
    Case 4
        curRemovable = 75
End Select
Select Case Monitor.Value
    Case 1
        curMonitor = 0
    Case 2
        curMonitor = 105
    Case 3
       curMonitor = 115
    Case 4
```

```
Lesson 11: Command Controls
```

```
curMonitor = 155
    Case 5
       curMonitor = 350
    Case 6
        curMonitor = 650
End Select
Select Case Printer.Value
    Case 1
        curPrinter = 0
    Case 2
       curPrinter = 150
    Case 3
       curPrinter = 650
    Case 4
       curPrinter = 995
    Case 5
       curPrinter = 350
    Case 6
       curPrinter = 450
End Select
Select Case NIC.Value
   Case 1
       curNIC = 0
    Case 2
       curNIC = 35
    Case 3
       curNIC = 105
End Select
If Fax Then
   curFax = 275
Else
   curFax = 0
End If
If Copier Then
   curCopier = 250
Else
   curCopier = 0
End If
If Scanner Then
   curScanner = 125
Else
   curScanner = 0
End If
Select Case OperatingSystem.Value
    Case 1
       curOS = 149.95
    Case 2
       curOS = 99.95
    Case 3
       curOS = 79.95
    Case 4
```

```
curOS = 299
    Case 5
        curOS = 319
End Select
Select Case Keyboard.Value
    Case 1
        curKeyboard = 14.95
    Case 2
        curKeyboard = 45
    Case 3
        curKeyboard = 109
End Select
If Speakers Then
   curSpeakers = 65
Else
   curSpeakers = 0
End If
If SurgeProtector Then
   curSurge = 24.95
Else
   curSurge = 0
End If
If AntiVirus Then
    curAntiVirus = 45.95
Else
    curAntiVirus = 0
End If
If Compressor Then
    curCompressor = 49.95
Else
   curCompressor = 0
End If
If ZIPDrive Then
    curZIPDrive = 140
Else
   curZIPDrive = 0
End If
If Modem Then
   curModem = 35.95
Else
   curModem = 0
End If
Select Case Mouse.Value
    Case 1
        curMouse = 19.95
    Case 2
        curMouse = 35
    Case 3
        curMouse = 45
End Select
```

Lesson 11: Command Controls

```
If MousePad Then
        curMousePad = 9.95
Else
        curMousePad = 0
End If
TotalOrder = curProcessor + curHardDrive + curRAM +
        curRemovable + curMonitor + curPrinter +
        curFax + curCopier + curScanner +
        curFax + curCopier + curScanner +
        curNIC + curOS + curKeyboard +
        curSpeakers + curSurge + curAntiVirus + curCompressor +
        curZIPDrive + curModem + curMouse + curMousePad
        txtTotalOrder = TotalOrder
End Sub
```

13. Now, every time the user makes a selection, we will call the central procedure to update its calculation. To achieve this, at the end of the previous procedure, enter the following (Normally, you should select a control in the Object combo box and, in the Procedure combo box, select AfterUpdate for a Group Box. The Click event should be selected by default for a check box. After selecting the event, call the above procedure. At the end, the event should look as follows (the result is the same if you simply type the following code):

```
Private Sub AntiVirus Click()
    CalculateOrder
End Sub
Private Sub Compressor Click()
    CalculateOrder
End Sub
Private Sub Copier Click()
    CalculateOrder
End Sub
Private Sub Fax Click()
    CalculateOrder
End Sub
Private Sub HardDrives Click()
   CalculateOrder
End Sub
Private Sub Keyboard AfterUpdate()
    CalculateOrder
End Sub
Private Sub Modem Click()
   CalculateOrder
End Sub
Private Sub Monitor AfterUpdate()
    CalculateOrder
End Sub
```

```
Private Sub Mouse AfterUpdate()
   CalculateOrder
End Sub
Private Sub MousePad Click()
    CalculateOrder
End Sub
Private Sub NIC AfterUpdate()
    CalculateOrder
End Sub
Private Sub OperatingSystem AfterUpdate()
   CalculateOrder
End Sub
Private Sub Printer AfterUpdate()
   CalculateOrder
End Sub
Private Sub Processors Click()
    CalculateOrder
End Sub
Private Sub RAM AfterUpdate()
   CalculateOrder
End Sub
Private Sub RemovableMedia AfterUpdate()
    CalculateOrder
End Sub
Private Sub Scanner Click()
   CalculateOrder
End Sub
Private Sub Speakers Click()
   CalculateOrder
End Sub
Private Sub SurgeProtector Click()
   CalculateOrder
End Sub
Private Sub ZIPDrive Click()
   CalculateOrder
End Sub
```

14. Close the Code Editor and preview the form

	Ocean Service Computer Store	e - Order Processing	
J	Order ID: 1		
	Primary Configuration Accessorie	s Operating System	Notes:
	Processors	Hard Drives	
	C AMD Athlon XP 1800+	40GB Ultra DMA	
	AMD Athlon XP 2000+	40GB UATA100 5400Rpm	
	C Intel Celeron 1.4GHz	C 60GB EIDE 7200Rpm	
	C Intel Pentium 4 2.0GHz	C 80GB EIDE 7200Rpm	
	C Intel Pentium 4 2.4GHz	C 100GB EIDE 7200Rpm	
	- Random Access Memory	Removable Media	
	C 128 MB	C 48X CD-ROM	
	256 MB	24X CDRW	
	C 512 MB	C 16X DVD	
	C 1.0 GB	C 32X DVD-CDRW Combo	
	Total Orde	er: 890.80	
Re	cord: 1	【 ▶★ of 1	

15. Save and close the form

Windows Controls: The Command Buttons

A button, sometimes referred to as a command button, is a rectangular object that allows the user to dismiss a dialog box or to initiate and action. This is done through a decision making process based on what a form or a dialog box is displaying. This could be an acknowledgement, such as a form displaying a simple message to the user. A user could also be asked to choose one button from a group of buttons on a form or a dialog box.

To use a button, the user position her mouse pointer on the desired button and presses the left button. This action is referred to as clicking.

Depending on how the button is implemented, it should be obvious to the user what to do with the button. A button can display a word or a group of words on its top. This word or group of words is referred to as the button's caption. The caption should be explicit enough to let the user know what the button is used for. A caption like OK usually means the user accepts what message the form or dialog box is displaying. A caption like Cancel is usually accompanied by another OK button on the form. When a button has a Cancel button, the user would usually click it as if saying, "Never Mind" or "I change my mind", etc.

Normally, to create a command button, you click the Command Button button on the Toolbox and click on the desired section of the form. On the Toolbox, if the Control Wizards button was down or clicked, a wizard would start to help you create a fully functional button. If you don't want to use the wizard, you click Cancel on the first page of the Command Button Wizard. Also, if you don't want to use the wizard, on the Toolbox, you can click the Control Wizards button to have it up. Microsoft Access, filled with limitless wizards, has greatly simplified the creation of command buttons. This is done using the Command Button Wizard. This wizard can help you configure almost any kind of behavior that a button is supposed to implement on a database application.

Practical Learning: Creating Command Buttons

- 1. Open the **Bethesda Car Rental3** database and click the Forms button on the Database Window
- 2. Double-click dlgAboutBCR to open



- 3. To switch it to Design View, right-click its title bar and click Form Design
- 4. On the Toolbox, make sure the Control Wizards button is selected Click the Command Button button
- 5. On the form, click under the Form Footer bar and on the right side of the Unbound text box
- 6. On the first page of the Command Button Wizard, in the Categories list, click Form Operations
- 7. In the Actions list, click Close Form

Command Button Wizard		
Sample:	What action do you want to happ Different actions are available for	en when the button is pressed? each category.
	Categories: Record Navigation Record Operations Form Operations Report Operations Application Miscellaneous	Actions: Apply Form Filter Close Form Edit Form Filter Open Form Print a Form Print Current Form Refresh Form Data
	Cancel < <u>B</u> ack	<u>N</u> ext > <u>F</u> inish

- 8. Click Next
- 9. On the second page, replace the content of the Text text box with OK (and make sure the Text radio button has been selected) and click Next
- 10. Change the name of the button to cmdClose and click Finish
- 11. Resize the button as follows:

digAboutBCR1a:F Form Header Detail Oetail Oetail Form Footer	Bethesda: Car: Rental(r) Release 2:02 This product is licensed to Bethesda: Car: Rental A Division of B CR: Productions, Inc. 1204 Mendelson Rid Suite DS Bethesda: MD 20854 Copyright (c) 2002 BCR Production	
Unbound		

12. Preview the dialog box:



13. Save the form. To close the it, click OK. Also close Microsoft Access

<u>Previous</u>	Copyright © 2002-2004 FunctionX, Inc.	<u>Next</u>



Application's Objects

A Database Application as a Multiple Document Interface

Most of your databases will consist of more than one object. For example, a typical database contains at least two forms and one or various reports. To make your database as easy to use as possible, when such a database contains various objects, you will typically allow the user to open more than one form or a form and a report at the same time. An application is called a multiple document interface (MDI) when the user can open more than one object or document inside of the application. For example, since you can open many forms in Microsoft Access and display them side by side, Microsoft Access qualifies as an MDI. To ease the user's experience with the various forms and reports of your database, you will usually allow the user to open more than one form or report at the same time, and you should allow the user to open a dependent object from the current object without using the Database Window or a switchboard.

The DoCmd Object

Microsoft Access provides an object that allows you to open one object from another. This object is called DoCmd. The DoCmd object can be used to open a form or report from another form. It is equipped with methods (also called functions; we will soon learn what a function is) that you can call to perform the desired operation. For example, to open a form, call the OpenForm using the following syntax:

DoCmd.OpenForm FormToOpen

The primary thing the OpenForm needs to know is which form you want to open. You can provide this as the name of the form. Since the name of an object is a string, you must provide it between double-quotes. Suppose that you have a form called Categories. If you want to open it, you can use code as;

DoCmd.OpenForm "Categories"

Previous

Copyright © 2002-2004 FunctionX, Inc.