The Super-Easy Guide to the Microsoft® Outlook® Object Model

White Paper

Published: March 1999

Table of Contents

Microsoft Office

The Super-Easy Guide to the Outlook Object Model

White Paper

Published: March 1999

For the latest information, please see http://www.microsoft.com/office/

Introduction

Customizing Outlook with the Outlook object model is easy. Really easy. You don't need a Ph.D. in Computer Science. You don't need to know C or C++ or any programming language. You don't need to know anything about object models. You don't even have to know how to program your VCR.

But don't take my word for it. Let me show you how easy it is. Take a look at these four lines of Outlook object model code:

```
Set NewMail = Outlook2000.CreateItem(olMailItem)
Set receiverOfMyMail =
newMail.Recipients.Add("everybody@thewholeworld.com")
NewMail.Subject = "The Outlook object model is Easy!"
NewMail.Send
```

Can you figure out what this code does in Outlook?

The Outlook object model uses language that for the most part is already familiar to you. For instance, **CreateItem** creates a new item in Outlook. **Add** adds something to an item in Outlook. **Subject** refers to the subject of an item. **Send** tells Outlook to send the item.

In these four lines of code, we create a new mail item, address it to "everybody@thewholeworld.com," set the subject to "The Outlook object model is Easy!," and send it.

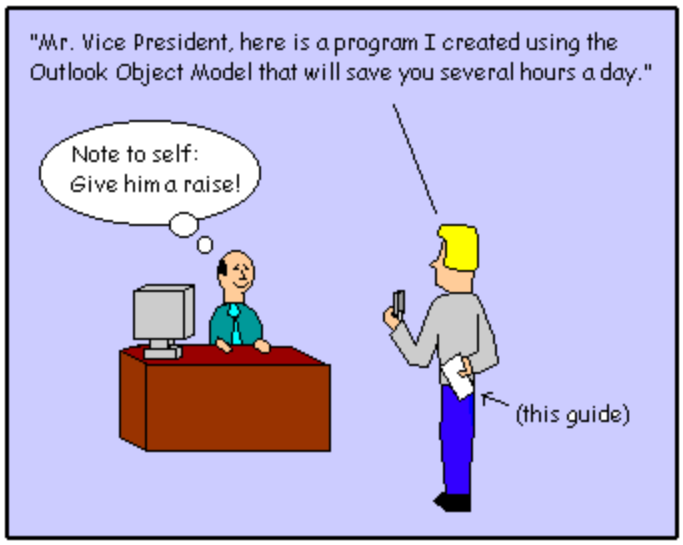
I told you it was easy. In just these four lines, you used Microsoft's messaging program to send an e-mail to the whole world declaring the benefits of the Outlook object model.

Why Should I Learn How to Use Outlook Object Model?

The Outlook object model allows you to customize Outlook to suit your organization's specific needs. The Outlook object model is great when someone in your organization demands (or if they're polite, requests) additional functionality in Outlook—and they need it right away.

For example: A vice president of your company storms into your office and says, "Every time I send a meeting request, I also need to inform certain people about the meeting without inviting them. Outlook won't let me do this!"

You know, however, that with just a little customization, Outlook will let your vice president do this. Next week, you walk into the vice president's office and hand him a solution to the problem, which you created yourself in a few hours using the Outlook object model. Of course, he doesn't have to know how easy it was.



This could be you! (individual results may vary)

Using This Guide

This guide is organized into eight lessons. Each lesson is hands-on—you will use the lessons with Outlook as you go. The guide is best used not as bed-time reading but as be-in-front-of-your-computer-and-try-it reading.

At the end of each lesson, there will be a "Hands-on Challenge ." You should try to complete each challenge—not only are they fun, but they will help you measure your understanding of the lessons. Answers and explanations are provided in the appendix.

What You Need to Know Before You Get Started

All you need is a working knowledge of Microsoft Windows and a familiarity with Microsoft Outlook. That's it.

If you already have programming experience, you can probably just glance at the "Programming Technique" sections, but the rest of the material may still be useful to you.

What You'll Know When You Finish the Lessons

After going through this guide and doing all the examples and exercises, you will be able to develop applications using the Outlook object model. You will also be able to discover on your own how to find the right tools in the Outlook object

model to solve a given problem. You will understand some key programming concepts, and you will gain a working knowledge of the Microsoft Visual Basic programming language.

In a nutshell, with the help of this guide, you will be able to apply the Outlook object model to meet your organization's needs.

Setting Up

To use this guide and go through the included examples, you need to have Outlook 2000. Because this guide uses Visual Basic for Applications (VBA), a new feature in Outlook 2000, the examples and lessons will not work with earlier versions of Outlook.

You do not need any special development tools to use the Outlook object model.

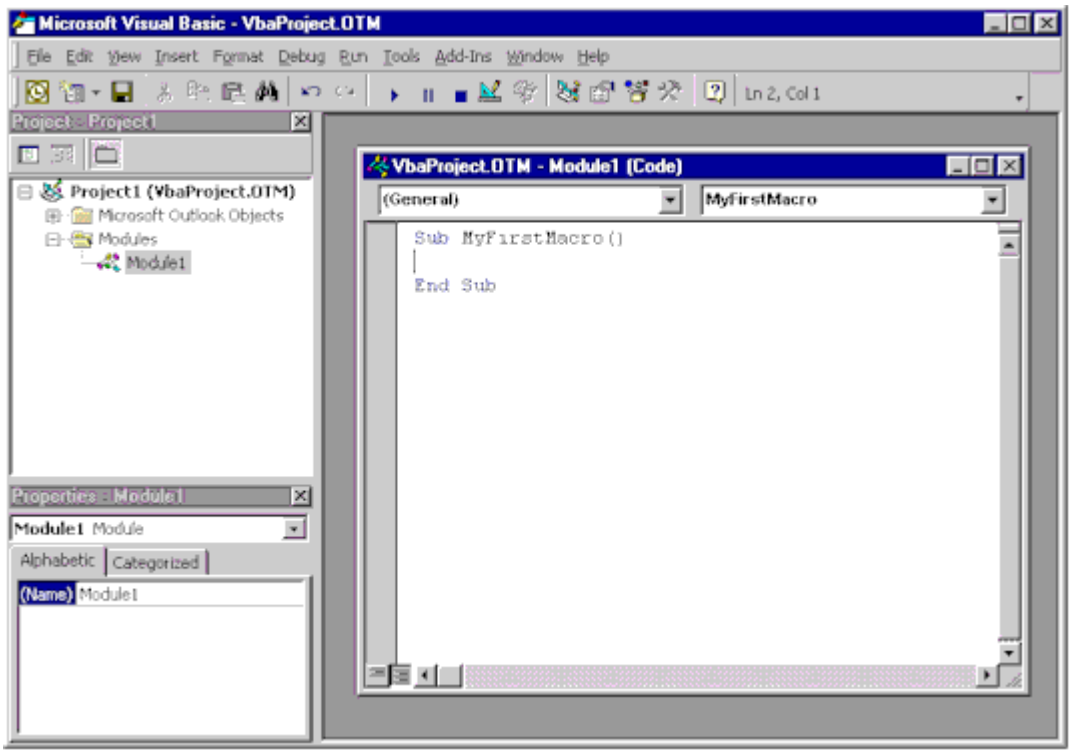
Lesson 1: Getting Started

We'll be using Visual Basic for Applications (VBA) for all the lessons in this guide. Visual Basic for Applications is a version of Microsoft Visual Basic integrated into Microsoft Office applications, including Microsoft Outlook. Solutions you create in VBA are called macros. A macro is a series of instructions in Visual Basic that perform something useful. When you write macros to perform tasks in Outlook, you write Visual Basic instructions that use the Outlook object model.

To start writing a macro:

1. Start Outlook 2000 (if it's not already running).
2. Point to **Macro** on the **Tools** menu, and then click **Macros.**
3. Now you need to come up with a name for your first macro. A macro name can't have a space in it, so type **MyFirstMacro**.
4. Click **Create**.

   Outlook will now automatically start the Visual Basic Editor, as shown in the illustration below:



The Microsoft Visual Basic Editor

By default, the Visual Basic Editor displays three windows:

- **Code window** (the big window on the right labeled something like "Project—Module1 (Code)"). This is where you write your Outlook macro code.
- **Project Explorer window** (top left, labeled Project_Project1). This window shows all the Microsoft Outlook Objects and modules available to you. (Don't worry if you don't know what Outlook objects are yet. That'll come later.) This window allows you to easily view and manage any number of VBA files.
- **Properties window** (bottom left, labeled Properties—Module1). This window displays the current set of properties for the selected item. Currently, the selected item is Module1.

Now let's use each of these three windows:

- "Module1" is such a dry and impersonal name, don't you agree? Let's change it. In the Properties window, go to the area where it says "(Name) Module1"—(Name) has a blue background. Select the text "Module1" and type something like "MyFirstModule." Notice that the name was changed in the Project window and in the title bar of the Module window!
- Now in the Project Explorer window, click the little plus sign next to "Microsoft Outlook Objects" to see what's inside. Currently, there's only one object, but it's a big one. It's called **ThisOutlookSession**. It's from this object that most of the fun happens.

- Go to the Code window for your module which you just renamed, "MyFirstModule." (If it's not already open, you can open it by double clicking the name, "MyFirstModule" in the Project Explorer Window.

To complete the macro:

1. Now let's enter some code in the MyFirstMacro routine. Enter the following:

```
Sub MyFirstMacro()
Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
NewMail.Subject = "(your name here) says: This really is easy! :)"
NewMail.Display
End Sub
```

   You don't have to know how or why this code works. You'll learn that in future lessons.

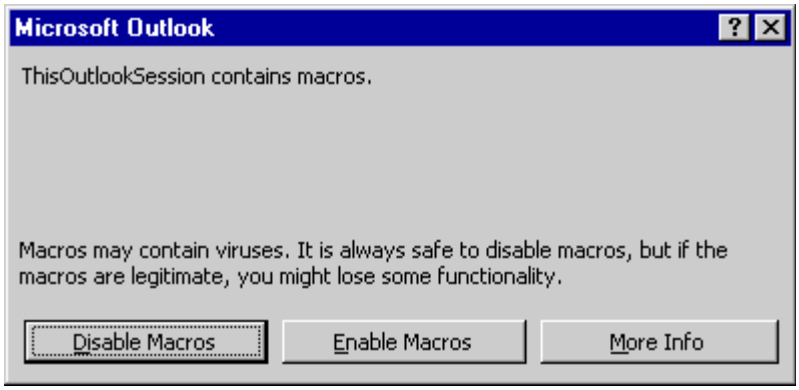
2. Now let's see it in action! First, close the Visual Basic Editor by clicking **Close and return to Microsoft Outlook** on the **File** menu.
3. In Outlook, point to **Macro** on the **Tools** menu and then click **Macros.**
4. The **Macros** dialog box will open with MyFirstMacro already selected. Click **Run**.

Did a mail message pop up with the subject you set using the Outlook object model? Congratulations! You're now an official Outlook object model programmer.

If an error dialog box came up, no big deal. Click the **Debug** button. Make sure the code looks exactly as is shown above and do steps 2-4 again.

About Security

Outlook will check whether or not you have any macros when you start Outlook. That is why you may see the following dialog box when you start Outlook:



When you see this dialog box, click the Enable Macros button. This will allow Outlook to run the macros you create. You can also choose to avoid this dialog box by reducing the level of security on your macros. To do so, point to **Macro** on the **Tools** menu and then click **Security.** Select the Low security option and click **OK**.

Hands-on Challenge #1

Try adding one line to the code to set the body of the message to "This is where the content of the message goes." (or any message you like). Hint: You will be using your "NewMail" message and the key word **Body**.

Lesson 2: Programming Concepts: Sub...End Sub and Procedures

So now that you've gotten your feet wet in this stuff, it's time to learn a bit about the water you're standing in. Let's take a close look at the code you just wrote:

```
Sub MyFirstMacro()
  Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
  NewMail.Subject = "(your name here) says: This really is easy! :)"
  NewMail.Display
End Sub
```

Let's first look at the Visual Basic key words **Sub** and **End Sub**. **Sub...End Sub** are used to begin and end a macro, following the pattern shown below:

```
Sub AnyNameHere()
  Some of that cool refreshing Object Model code here
End Sub
```

AnyNameHere is the name of the name of a macro or *procedure*. A procedure is a small set of code that you create that does something. MyFirstMacro is an example of a procedure. A procedure doesn't have to be a macro, however. You can create a procedure and then "call" that same procedure from another procedure, as in the following example:

```
Sub MyFirstMacro()
  MakingMail
End Sub

Sub MakingMail()
  Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
  NewMail.Subject = "Hey, this Sub/End Sub thing is easy too!"
  NewMail.Display
End Sub
```

If you run MyFirstMacro, the procedure MakingMail will run and you will see a mail message with the subject, "Hey, this Sub/End Sub thing is easy too!"

So why would you want to do this? Creating separate procedures allows you to organize your code in a nice, clean way, and it allows you to do common procedures easily. For example, say we needed to create and display three different mail items with the subject, "Hey, this Sub/End Sub this is easy too!," you could do that by changing the code in MyFirstMacro as follows:

```
Sub MyFirstMacro()
  MakingMail
  MakingMail
  MakingMail
End Sub
```

Hands-on Challenge #2

Edit your MyFirstMacro macro in the Visual Basic Editor and change the code to use a new procedure such as the MakingMail procedure to create a new piece of mail five times.

Lesson 3: Programming Concepts: Objects, Methods, and Properties

At some point, you may have heard all the hoopla over "object oriented" programming. Object oriented programming is the key concept behind C++ and Java, the most widely used programming languages today. What you probably didn't know, however, is that just by finishing lessons 1 and 2, you can now call yourself an object oriented programmer!

That's right. The Outlook object model uses object oriented programming. Fortunately, to use and understand the Outlook object model, you don't need to take a class in the subject or write a thesis on it. To gain a working knowledge of the Outlook object model, you only need to know three concepts:

Concept	Description	Example
Object	A "thing"	Mail item
Method	Something a "thing" can do	Show itself to the user
Property	A characteristic of a "thing"	Subject

Everyday things can be thought of as objects, methods, and properties. For instance, consider a car as an object. A car object has methods, which are various things it can do, such as Drive, Start, Turn Left, Turn Right. A car also has properties that describe it: the color is beige and the number of headlight is two.

"I am an object. As a car object, I have methods which are things I can do: Drive, Start, Turn Left, Turn Right. I also have properties which are characteristics that describe me: My color is beige and the number of headlight I have is two."

Let's take a closer look at the code you've already written and see where the objects, methods, and properties are:

```
Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
NewMail.Subject = "Hey, this Sub/End Sub thing is easy too!"
NewMail.Display
```

There are two *objects* in this code. The first is **NewMail**. The second is **ThisOutlookSession**. It's easy to visualize both a mail item and a session of

Outlook as "things." Incidentally, the Outlook object model is simply a list of the objects that we can use to program Outlook.

Whenever you first use an object, you have to use the **Set** key word. Objects take up memory in the computer; the **Set** key word allocates the memory required for an object.

You can give objects any name you want. In the above example, I gave the mail item object the name, **NewMail,** but you can change the name to suit your mood. The object, **ThisOutlookSession** was created for us—you can see it in the Project Window under Microsoft Outlook Objects.

The MyFirstMacro code contains two *methods*. The first is **CreateItem**. The second is **Display**. A method is always associated with an object. In this case, **CreateItem** is associated with the object, **ThisOutlookSession**. To use a method, you simply place a period in between the object and the method. For example, `NewMail.Display`.

As described above, a method is "Something a thing can do." In the example you've written, an Outlook Session can create a piece of mail—**CreateItem**. And a piece of mail can display itself to the user—**Display**.

Sometimes methods need additional information. For example, the **CreateItem** method needs to know what type of item to create. We tell it to create a mail item by including that information, **olMailItem**, in parenthesis after the name of the method. Some methods require more than one piece of information, while others, such as **Display**, require none.

The MyFirstMacro code contains one *property*. It is **Subject**. Like a method, a property is always associated with an object. In this case, **Subject** is associated with the object, **NewMail**. To use a property, you simply place a period between the object and the property. For example, `NewMail.Subject`.

Once again, think of a property as "A characteristic of a thing." For instance, the subject is a characteristic of a mail item. Another example: The start time is a characteristic of an appointment item.

One last thing to know: Every object is of a specific type. Each type of object has its own set of methods and properties. In the above example, **NewMail** is a "mail item" object. Mail items have methods, such as **Display**, and properties, such as **Subject**, that other types of objects do not have. For example, the following instruction:

```
ThisOutlookSession.Subject = "Hello"
```

would not work because objects of type "Outlook Session" do not have properties called **Subject**.

Hands-on Challenge #3

Look at the following make-believe Object model code and determine which parts are objects, methods, and properties (there are three of each):

```
Set PetStore = ShoppingMall.GetStore(aPetStore)
PetStore.OpeningTime = 9 AM
Set Dog = PetStore.GetPet(aDog)
Dog.Breed = "Cocker Spaniel"
Dog.Color = "Blond"
Dog.WagTail
```

Lesson 4: Real-World Example #1

Suppose that each day, you need to order lunch for the people in your group. There are a few restaurants to order from, but you only want to order from one. The best way to choose the restaurant is to take a vote. Let's consider how you would do this as a user in Outlook:

1. Create a new mail item.
2. Click **Options** on the **View** menu.
3. Select **Use voting buttons**
4. Enter the names of the restaurants, each separated by a semicolon. For example: "Taco Temple; Burger Palace; Chicken Chums"
5. Close the dialog box.
6. Address the mail to your co-workers.
7. Enter in the subject and text in the body explaining to your co-workers what the mail is for.
8. Send it.

The good news is that Outlook provides a good way to poll many people. Your co-workers will be able to choose by simply clicking on a button with the name of the restaurant they want.

The bad news is that you'll have to do the same above eight steps every morning at work. This is where the Outlook object model can help: We'll create a macro to turn those eight steps into zero steps.

Let's jump right in and create this macro:

To create the PollRestaurant macro

1. Create a new module called PollRestaurant. In Outlook, point to **Macro** on the **Tools** menu, and then click **Macros.** Type the word **PollRestaurant** and click **Create**. The Visual Basic Editor starts up with your new PollRestaurant macro ready to fill in.
2. Type the following code:

```
Sub PollRestaurant()
   Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
   NewMail.Subject = "Please open and make your vote for lunch!"
   NewMail.Body = "Click one of the buttons above to make your" &_
                  "vote. I'll be tallying at 11 AM. " &_
                  "Lunch will be in the break room at noon."
   NewMail.VotingOptions = "Taco Temple; Burger Palace; " &_
             "Chicken Chums; Pasta Hut"
   Set receiverOfMyMail =
NewMail.Recipients.Add("hungrypeople@mycompany.com")
   NewMail.Send
End Sub
```

   Note: hungrypeople@mycompany.com is a distribution list of people who will be having lunch.

3. Run your macro to see it in action. You will send out a mail message to the distribution list with the voting options you selected.

Note: Because you are not displaying the item, you will not see anything. Check to see if the message got sent by checking your Sent Items folder.

That's it! You're done! And to do this, we only introduced one new line of code:

```
NewMail.VotingOptions = "Taco Temple; Burger Palace; " &_
                "Chicken Chums; Pasta Hut"
```

VotingOptions is a property of the NewMail object. We set the VotingOptions to be a series of restaurants separated by a semi-colon. (The same thing you would do if you added the voting options to the message manually.

Using Events

Now you have a nice little macro that sends out a piece of mail asking your employees of their lunch choice. But you still have to actually run the macro for it to work. Now say, for instance, the first thing you do when you get to work in the morning is to start up Outlook. And let's say that the last thing you do before you leave work is to quit Outlook. Wouldn't it be nice to have Outlook automatically run your macro every time Outlook starts up? Then you could send out the lunch polling mail every morning without doing any work at all!

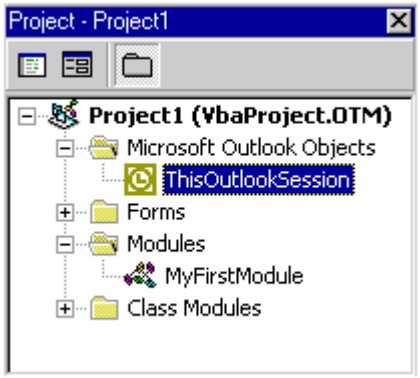
You can do that by using an object *event*. An event occurs when something happens to an object. For instance, when you receive mail, a NewMail event happens to the **ThisOutlookSession** object. When you start up Outlook, a StartUp event happens to the **ThisOutlookSession** object.

To illustrate further, think back to the car analogy. You might want the car object to do something when someone puts a key in the ignition. In this case, you would use a car event called something like DriverPutKeyInIgnition. When that event gets called, you would want to start up the car's engine and get ready to go.

"There are several events I respond to. These include turning the key in my ignition, shifting to a different gear, pressing on the gas pedal, and pressing on the break pedal. I have to do the right thing whenever a driver does one of these events."

To run your macro each time Outlook starts up

1. Start up the Visual Basic Editor (if it isn't already open).
2. Click on the "+" sign of the Microsoft Outlook Object item in the Project Explorer Window. You should see an item under it labeled "ThisOutlookSession."



3. Double-click on "ThisOutlookSession." A Code window should appear with the title, "ThisOutlookSession."
4. Notice two drop down list under the title bar of the Code window. The left drop down list has a list of objects associated with **ThisOutlookSession**. Click there and select the object **Application**.

5. The right drop down list has a list of macros and events associated with the item selected in the left drop down list. Click on the right drop down list and select "Startup." Startup is the event you will be using. (Note just for fun the other events you can choose from.)
6. A Startup procedure will appear in the Code window. To call your macro, simply type the name of the macro in the procedure. Example:

```
Private Sub Application_Startup()
      PollRestaurant
End Sub
```

That's it! You're done! Quit Outlook and start it up again to see if it worked.

Hands-on Challenge #4

Create a new macro that polls your co-workers for their breakfast food choice. Because you'll want people to vote at the end of the day (so you can order breakfast first thing in the morning), have the macro called every time you quit Outlook.

Lesson 5: Creating Your Own Dialog Boxes

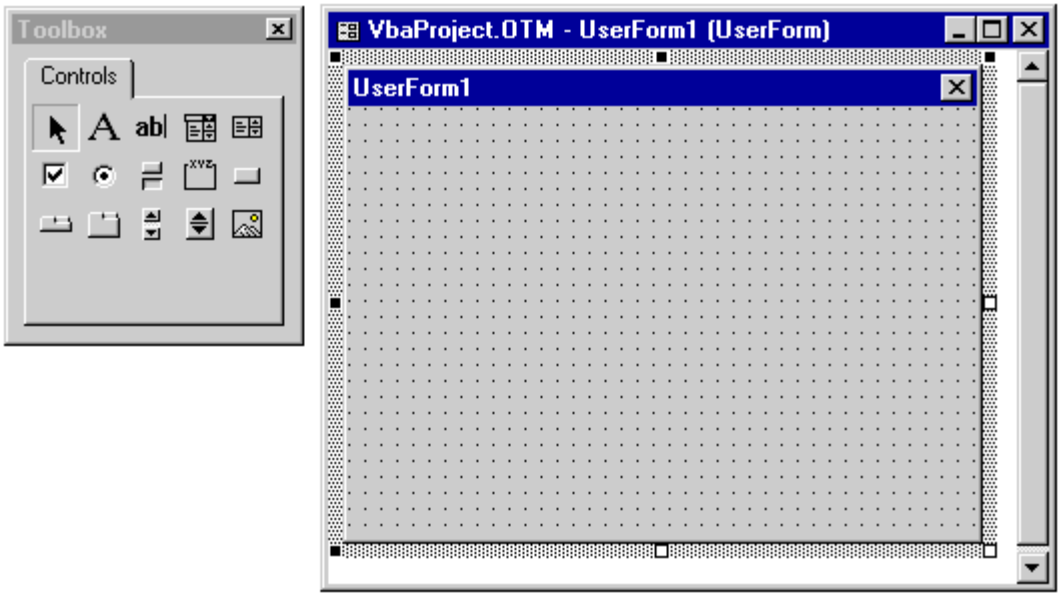
Now for one of the coolest parts of VBA—forms. A form is a dialog box-like object that you can create and design in VBA. You can add checkboxes, text, pictures, and all sorts of Windows features to your form without writing a single line of code.

Why do you need forms? Sometimes your macro will need to communicate information to users or get information from users. For instance, you may want to show a progress dialog box while your macro is running, or you may want to let users set some options before your macro does its magic.

To demonstrate, let's go back to the PollRestaurant macro you created in Lesson 4. The PollRestaurant macro sends out a piece of mail with voting options every time you start up Outlook. But what if you start up Outlook at different points throughout the day? What if you go to the office on the weekend and start up Outlook? You certainly wouldn't want to flood your co-workers e-mail boxes with "What do you want for lunch?" e-mail five times a day. The solution is to present yourself with *the option* to send out the mail every time you start Outlook.

To create the "Lunch for Co-workers" dialog box

1. In Outlook, point to **Macro** on the **Tools** menu, and then click **Macros.**
2. Let's create a new macro. Type the name "PollRestaurentPrompt" and click **Create**.
3. Now you should be in the Visual Basic Editor. Click **UserForm** on the **Insert** menu.

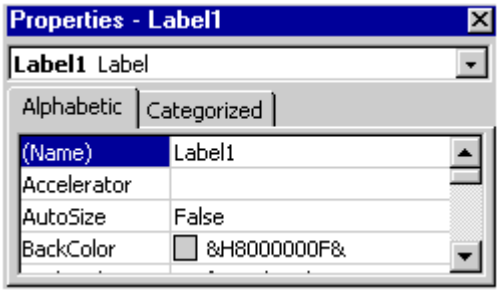


Your form will appear along with a toolbox of user controls. The form is where you design dialog box that the user will see and interact with.

4. Click on the form to select it. Look at the Properties window (the one at the lower left.) Change the (Name) property to "PromptDialog." Press Return.
5. Change the Caption property to "Lunch for your co-workers?" Press Return. Notice that the title of your form will be what you just typed.
6. Hover over the different user controls in the toolbox and look at the screentips to see what each of them is.

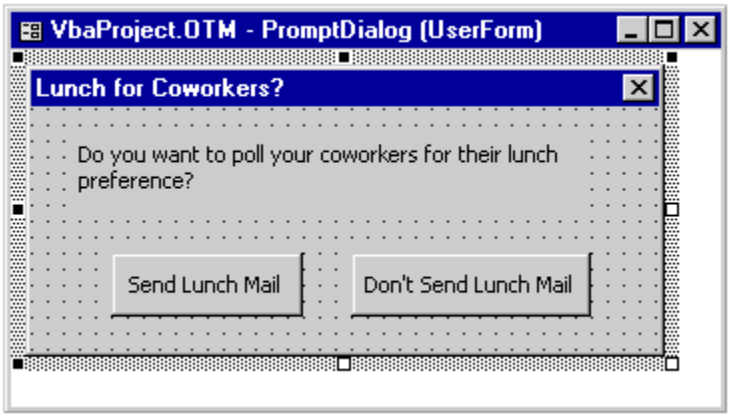
   Before we continue, let's consider for a moment what we'll need for our PollRestaurantPrompt macro. We want to give the user a choice to send or not to send the mail with the restaurant voting options. A simple way to do this is to add some explanatory text and then add two buttons, "Send Lunch Mail" and "Don't Send Lunch Mail"
7. Select the object in the toolbox marked **Label**—it's the graphic with the capital A.
8. Go to the form in the PromptDialog window. Click and drag the mouse to select the area where your explanatory text will be. (Don't worry, you can always move and resize it after you create it.)

   Note: The label you just created is an object. It has methods and properties just like a mail item or an Outlook session.
9. Look at the Properties window (on the lower left) for list of the label's properties. The first few you should see are (**Name**), **Accelerator**, and **AutoSize**.



10. Click on the value for the **Caption** property (it's probably something like "Label1"). Type a new value, for example, "Do you want to poll your co-workers for their lunch preference?"
11. Press Return. You should see in the Module Window that the label shows what you just typed.
12. Now you need to create your two buttons. Click the form to show the Toolbox again. Select the object marked "CommandButton"—it's the simple rectangle graphic.
13. Just like you did with the label in step #8, create a button in your form.
14. To create a second button, click on the **CommandButton** object in the Toolbox and drag another button onto your form.
15. Select one button and set some properties in the properties window. Set the (Name) to be "Yes." Set the Caption to be "Send Lunch Mail."

16. Select the other button and set the properties in the properties window. Set the (**Name**) to be "No." Set the **Caption** to be "Don't Send Lunch Mail."



Making Your Dialog Box Work

So now you have this cool dialog box. But it doesn't do anything. To make it work, you only need four lines of Outlook object model code. The tricky thing is that you need these lines in three places. Here's how you do it:

1. The first thing we want to do is show this dialog box when the user starts Outlook. Go to the ThisOutlookSession project window (by double-clicking on ThisOutlookSession under Microsoft Outlook Objects in the Project Window.)
2. Go to the Application_Startup event and replace the line "PollRestaurant" with "PromptDialog.Show." The code should look like this:

```
Private Sub Application_Startup()
  PromptDialog.Show
End Sub
```

   Note: **PromptDialog** is an object. **Show** is a method of that object.

3. Close the ThisOutlookSession Code Window. Open the PromptDialog form. (If it's not already open, you can open it by double-clicking on PromptDialog under Forms in the Project Explorer window.)
4. Now we want to do the right thing depending on which button the user presses. Double-click on the button labeled, "Send Lunch Mail."
5. The Code Window will open with the code for PromptDialog. VBA will create the event that is called when the user presses the button. When the user presses this button, we want to do two things. We want to send out the lunch polling mail and we want to hide the dialog box. (We assume you have already created the PossRestaurant macro from Lesson 4.) The code should look like this:

```
Private Sub Yes_Click()
  PollRestaurant
  PromptDialog.Hide
```

```
End Sub
```

Note: **PromptDialog** is an object (**PromptDialog** is the name you set in the Properties window for the dialog box.) **Hide** is a method of that object.

6. Close that window. Now double-click on the button labeled, "Don't Send Lunch Mail." In that procedure, you only need to do one thing. Hide the dialog box. The code should look like this:

```
Private Sub No_Click()
  PromptDialog.Hide
End Sub
```

That's it! You're done! Close the Visual Basic editor, close Outlook and restart to see your macro and your dialog box in action.

Hands-on Challenge #5

Create a third button on your dialog box that gives the user the option to send out mail asking for your co-workers' breakfast choice. Use the macro you created from Hands-on Challenge #4. Have this dialog box appear both when you start and when you quit Outlook.

Lesson 6: Programming Concept: Collections

So far, you have learned about objects, properties, methods, and events. There is one last category you need to understand to fully use the Outlook object model: Collections.

A collection is a special type of object—an object that is a group of other objects. So, for instance, if "car" is an object, "cars" is a collection, a collection of cars.

A collection can also be a property of another object. Continuing with the car example, "doors" can be a property of a "car" object as well as a collection of "door" objects. Therefore, we can understand the relationship like this:

"Cars" is a collection of "car" objects. Each "car" object has a property called "doors." "Doors" is a collection of "door" objects.

"We are a collection of car objects. Each of us has a property called doors. Each of these properties is a collection of door objects."

Collections in Outlook

The Outlook object model has many types of collections. There is an example of one on the very first example of this guide:

```
Set receiverOfMyMail =
newMail.Recipients.Add("everybody@thewholeworld.com")
```

Recipients is a collection of **Recipient** objects. **Recipients** is also a property of the **NewMail** object. It's easy to see the usefulness of collections. A piece of mail can be addressed to any number of names. The **Recipients** collection gives us an easy way to manage these names.

Other collection objects in Outlook include:

- **Folders**—A collection of **Folder** objects.
- **Items**—A collection of **Item** objects (such as a mail message, appointment, etc.)
- **Attachments**—A collection of **Attachment** objects.

Methods and Properties of Collections

All collections have methods and properties that allow you to access the individual objects in the collections. Three of the most important methods and properties will be covered here.

- **Count** property—This property tells us how many individual objects are in a collections. For example:

```
Dim numberOfRecipients
numberOfRecipients = newMail.Recipients.Count
```

In this example, `numberOfRecipients` is a variable. A variable is something we define that allows us to record information and use it later. After this code runs, the `numberOfRecipients` variable will be equal to the number of recipients in the **NewMail** item.

Before we use a variable, we have to define it. We define it by using the key word **Dim**. We can call our variables anything we like. For instance, this code would work just the same:

```
Dim myDogBitMe
myDogBitMe = newMail.Recipients.Count
```

Note: A collection can be empty, in which case the **Count** property is zero.

- **Item** method—This method allows you to access a specific object in a collection. For example:

```
Set aRecipient = newMail.Recipients.Item(2)
```

The number in the parenthesis indicates which recipient you want to access. For this example, the object **aRecipient** will become the second **Recipient** object in the **Recipients** collection.

You can also use a variable in the parenthesis. For example:

```
Dim numberOfRecipients
numberOfRecipients = newMail.Recipients.Count
Set theLastRecipient = newMail.Recipients.Item(numberOfRecipients)
```

Here, you are first setting the variable `numberOfRecipients` to be the number of **Recipient** objects in the **Recipients** collection. Then you are accessing the last recipient. So if there are five recipients, the last item can be specified by the number 5. The last line will be the same as saying:

```
Set theLastRecipient = newMail.Recipients.Item(5)
```

- **Add** method—This method allows us to add additional objects to a collection. For example:

```
Set receiverOfMyMail = newMail.Recipients.Add("mom@family.com")
```

  How you use the **Add** method might vary depending on which collection you're adding to. In most cases, like the one above, you will at least have to specify a name for the new object (like mom@family.com).

One more note about variables: Because a variable records information, its value only changes when you change it. As an analogy, think of a variable as an audio tape:

Let's say we create an audio tape called "MyCurrentAge." On this audio tape, we record your current age. Now let's say we listen to the audio tape five years from now. What will you hear when you play back the audio tape? You will hear, not your current age, but your age five years ago. This is an important concept and you can use it to correctly answer Hands-on Challenge #6.

Hands-on Challenge #6

Look at the following make-believe Visual Basic code and answer the questions that follow. Assume that at the start, **Papers** collection is empty (its **Count** property is equal to zero.) Hint: Step through the code line by line and keep track of the `numberOfPapers` variable and the objects in the **Papers** collection.

```
Dim numberOfPapers
Set paper1 = MyDesk.Papers.Add("The Foofle Report")
Set paper2 = MyDesk.Papers.Add("The Mooble Report")
currentNumber = MyDesk.Papers.Count
Set paper3 = MyDesk.Papers.Add("The Garble Report")
Set paper4 = MyDesk.Papers.Item(currentNumber)
Set paper5 = MyDesk.Papers.Item(1)
Set paper6 = MyDesk.Papers.Item(currentNumber + 1)
```

1. What is the value of the variable, `currentNumber`?
2. What is the value of `paper3`?
3. What is the value of `paper4`?
4. What is the value of `paper5`?
5. What is the value of `paper6`?

Lesson 7: Programming Concept: If This, Then That

Sometimes we need to control which code is used based on the state of things. For instance, in the PollRestaurant macro, it would make sense to send out lunch mail, but only if the time is not past 1 PM. Or we may want to create a macro that runs only when the user sends out mail to certain people or even a specific number of people.

We can create procedures that respond to different conditions by using the **If...Then** control statement. The **If...Then** control statement is one of many

Visual Basic tools that can direct the flow of your code. The format of the **If...Then** control statement is as follows:

```
If <expression> Then
<code here>
End If
```

In the above code, <expression> represents something that can be True or False. <code here> represents the code that will run if <expression> is determined to be True.

<expression> examples:

`Papers.Count = 3`	True if there are three Paper objects in the Papers collection.
`numRecipients > 0`	True if the numRecipients variable is greater than 0.
`numRecipients <> 5`	True if the numRecipients variable does not equal 5.

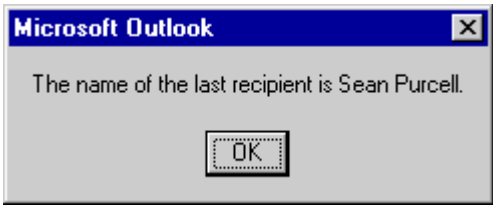
The following two examples help demonstrate how the **If...Then** control statement can be used.

If...Then Example 1: If this mail is addressed to anyone, then...

```
Dim numberOfRecipients
numberOfRecipients = myItem.Recipients.Count
If numberOfRecipients > 0 Then
      Set myRecipient = myItem.Recipients.Item(numberOfRecipients)
      MsgBox "The name of the last recipient is " &_
             myRecipient.Name & "."
End If
If numberOfRecipients = 0 Then
      MsgBox "There are no recipients in this item."
End If
```

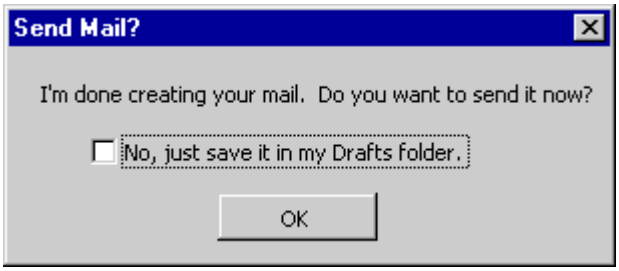
In this example, we display a dialog box that tells the user the name of the last recipient in the mail message. If the mail message has no recipients, we tell the user that.

Note: **MsgBox** is a Visual Basic statement used to display simple messages. You can display a single string or any number of strings joined by a "&" sign. In the above example, if the name of the last person the mail is addressed to is "Sean Purcell," the user would see a dialog box like this:



If...Then Example 2: If this option is checked, then...

If we present the user with a dialog box like this:



When the user clicks the **OK** button, we have to run different code depending on whether the user checked the checkbox labeled "Just save it in my Drafts folder." If it's not checked, we'll send the message. If it is checked, we'll just save it so the user can send it later.

To understand how this would work, first you need to know that a checkbox is an object you can draw right onto a form—just like a button or a label like you created in Lesson #5. Because it is an object, it has properties. One of those properties is called "Value." The Value property can either be True (it's checked) or False (it's not checked.)

The code needs to run when the user clicks the OK button. So we add the following code to Click event of the OK button. In this example, we'll assume that the name of the form is **PromptDialog**, the name of the button is **OK**, and the name of the checkbox is **justSendDraft**.

```
Private Sub OK_Click()
   If PromptDialog.justSendDraft.Value = True Then
     MyItem.Save
   End If
   If PromptDialog.justSendDraft.Value = False Then
     MyItem.Send
   End If
   PromptDialog.Hide
End Sub
```

If the user selects the check box, Outlook now saves the message to the user's Drafts folder. If the user does not select the check box, Outlook sends the message.

Hands-on Challenge #7

Make one addition to the PollRestaurant macro you created in lesson 4 and added a dialog box to in lesson 5. In the dialog box, add a checkbox that allows the user to edit the polling mail before sending it. The checkbox can have a Caption like, "Edit the polling mail before sending." If this is checked, don't send the mail—just create the mail and display it to the user.

Hint: You may want to create another PollRestaurant macro (called PollRestaurant2) that displays the message by using the Display method of the Mail Item object. Then when the user clicks the "Send Lunch Mail" option, you would call one macro or the other depending on the user's choice.

Lesson 8: Real World Example #2

Suppose you're an assistant to an executive in a large corporation. One part of your job is sending out important e-mail to groups in your company. There are about five different groups (distribution lists) you frequently have to send e-mail to. Sometimes you'll want to address an important e-mail to all 5 distribution lists. Sometimes you'll want to address an important e-mail to just one or two.

Because you send these e-mails out many times each week, you want to make it super-easy for you to select which of these distribution lists you (or your manager) want to send the mail to. You don't want to have to go through your company's Global Address List (which has too many names to count) and you don't want to have to type each name or find each name from a list individually.

How do you make this task super-easy? By using the Outlook object model to create a dialog box where you can just check which of these groups you want to send your mail to!

For the purpose of this example, we'll assume your company makes wacky musical instruments. The names of the distribution lists that interest you (with descriptions) are:

Blue Horn Sales Division	All the sales personnel responsible for selling blue horns.
Gold Flute Sales Division	All the sales personnel responsible for selling goldflutes.
Flute And Horn Marketing	All the people responsible for marketing flutes and horns.
Blue Horn Design Team	All the engineers responsible for designing horns.
Gold Flute Design Team	All the engineers responsible for designing flutes.

Step 1: Create a Contact

Before we begin, let's take a step back and imagine how you will use this name-select dialog box. A simple way would be to have the dialog box show up when you send your mail. At that point, you can check off which of the above distribution lists you want to send the mail to. When you click OK, Outlook will add the appropriate names and send the mail.

But wait. You don't always want to see this dialog box every time you send mail. You only want to see this dialog box when you need it. You can do this by first checking who the mail is addressed to. If the first address is a specific name, you will show the dialog box. If the first address is not that specific name, you will not show the dialog box.

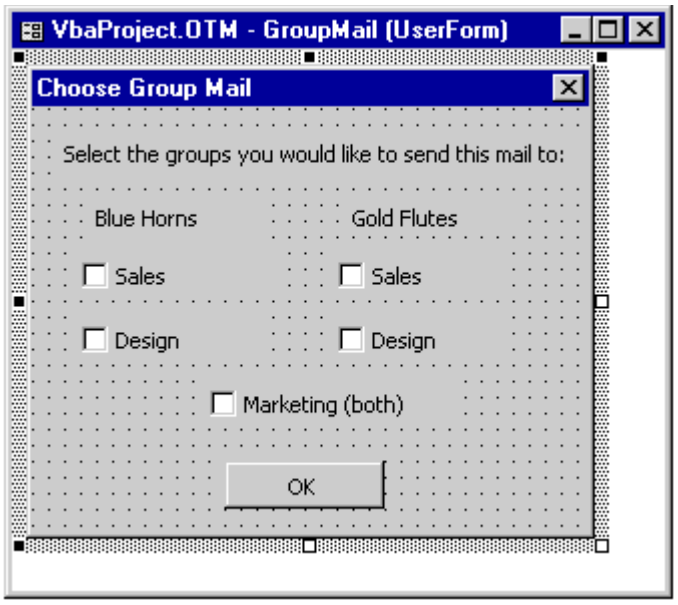
So to get started, first you need to create that contact:

1. Go to the **Contacts** folder in Outlook and create a new contact.
2. Set the name of the Contact to be something simple like "Group Mail." Whenever you want to send out mail to the above distribution lists, you will use this name.
3. Set the e-mail address to be your e-mail address. (You won't actually be sending mail to this contact, so it doesn't really matter what the e-mail address is, as long as it's there.)
4. Save and Close the contact.

Step 2: Create the Dialog Box

Now we need to create the actual dialog box the user will see when sending mail to the "Group Mail" contact:

1. In the Visual Basic Editor, click **UserForm** on the **Insert** menu.
2. Now you will have a new dialog box to add your own things to.
3. Add 3 labels, 5 check boxes, and 1 button so your form looks like this:



   Remember, the **Caption** property sets the text the user sees in the dialog box.

4. For the check boxes, the button and the form itself, set the **Name** property to be descriptive. Examples:

Control	Name
The form	GroupMail
OK Button	OK
Blue Horns Sales checkbox	BlueHornSales
Gold Flute Design checkbox	GoldFluteDesign

5. (Bonus item!) For each of the check boxes, set the **ControlTipText** to be the description of each group. This is how you set the screen tips the user sees when hovering over one of your controls.

Step 3: Make It Work

To make the program work, you will need to write Outlook object model code for two events: When the user sends mail, and when the user clicks the **OK** button of your dialog box. Since we just created the dialog box, let's write the **OK** button code first:

1. From the form in the Visual Basic Editor, double click on the OK button. Add one line to the OK_Click procedure to hide the dialog box:

```
Private Sub OK_Click()
      GroupMail.Hide
End Sub
```

   Note: GroupMail is the name of the form and OK is the name of the OK button.

2. Now go to the Outlook event that is called when you send mail. Go to the **ThisOutlookSession** module (double click on **ThisOutlookSession** under Microsoft Outlook Objects in the Project Window.)

3. In the Application_ItemSend procedure, clear any existing Outlook object model code (from the PollRestaurant macro you did in previous lessons.)

4. We will be entering the code in three steps. In the first step, we will make sure that the mail is addressed to at least one e-mail alias:

```
Private Sub Application_ItemSend(ByVal Item As Object, Cancel As
Boolean)
      If (Item.Recipients.Count > 0) Then
            Set firstRecipient = Item.Recipients.Item(1)
      End If
      Cancel = True
End Sub
```

   The If statement tells us whether the number of recipients is greater than 0. If it is greater than zero, we will continue on with our code.

   Note that if the mail is not addressed to anyone, the next line,

```
Set firstRecipient = Item.Recipients.Item(1)
```

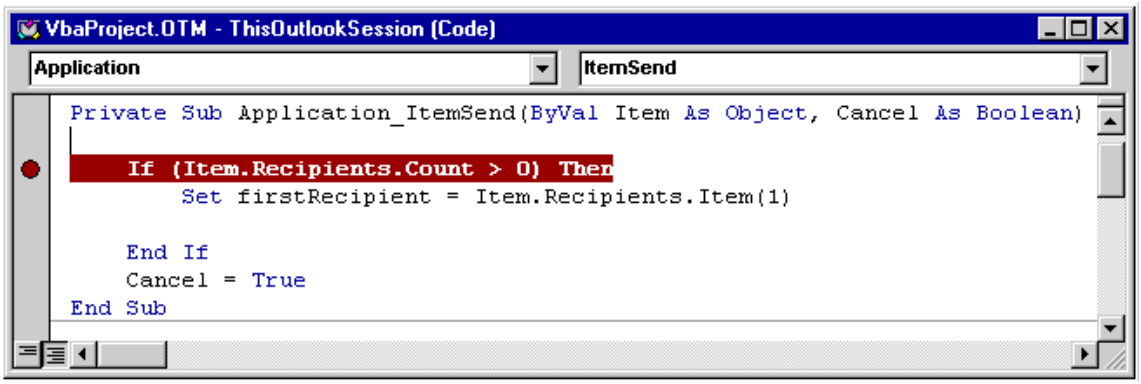
   would not work because there would be no first recipient. That is why it is important to make sure the mail is addressed to at least one e-mail alias.

Intermission—Using the VBA Debugger

Before we finish, now is a good time to introduce you to the Visual Basic Editor Debugger. The Debugger is used to help you see how your code works and to fix any problems that might exist in your code.

To use the Debugger, you create a "Breakpoint" in your code. Create a breakpoint by clicking on the light verticle bar to the left of your code. Here you can add and remove breakpoints (they look like little Stop signs.)

As an example, create a breakpoint at the first If statement:



Now when your code runs, your code will be stopped at this line. From there, you can "step" through your code by pressing the F8 key. When you just want your code to continue running, press F5.

Try it! With your breakpoint at the first If statement, create a piece of mail in Outlook and don't address it to anyone. Send the mail. VBA will open stopped to the line where you set the breakpoint. Hit F8. It should skip all the code within the If statement. Now create another piece of mail with recipients. Send the mail. Now VBA should go to the code within the If statement. Cool, huh?

Step 4: Finishing the Code

Let's continue making this work:

1. Add the following 5 lines of code within the first If statement:

```
Private Sub Application_ItemSend(ByVal Item As Object, Cancel As
Boolean)
If (Item.Recipients.Count > 0) Then
Set firstRecipient = Item.Recipients.Item(1)
   If StrComp("Group Mail (E-mail)", firstRecipient.Name,
vbBinaryCompare) = 0 Then
      Item.Recipients.Remove 1
      GroupMail.Show
   End If
   Cancel = True
End If
End Sub
```

After we set the **firstRecipient** object to be the first recipient in the mail message, we use a Visual Basic function, **StrComp**, to see if that recipient is the same as the placeholder contact we created at the beginning of this lesson. (Note that Outlook appends the names of personal contacts with "(E-mail).") If StrComp(string1, string2, type of comparison) = 0, that tells us that string1 and string2 are identical.

If the first recipient is our "Group Mail (E-mail)" contact, we then remove that contact from the mail. We do that via the Remove method which is common to all collections. The "1" tells VBA to remove the first recipient in the collection.

Then we show the user the GroupMail dialog box.

Use the Visual Basic Editor Debugger to make sure this works properly with mail addressed and not addressed to the Group Mail contact.

We put the line, Cancel = True, so that we don't actually send the mail message when this procedure is done. At the end, you may want to take this line out, but it helps to be able to use the same mail (and not send out a bunch of bogus mail) until we're done.

2. Finally, add code after you show the GroupMail dialog box which behaves based on the check boxes the user set:

```
Private Sub Application_ItemSend(ByVal Item As Object, Cancel As
Boolean)
If (Item.Recipients.Count > 0) Then
  Set firstRecipient = Item.Recipients.Item(1)
  If StrComp("Group Mail (E-mail)", firstRecipient.Name,
vbBinaryCompare) = 0 Then
     GroupMail.Show
     Item.Recipients.Remove 1
     If GroupMail.GoldFluteDesign.Value = True Then
       Set newRecipient = Item.Recipients.Add("Gold Flute Design
Team")
       newRecipient.Resolve
     End If
     If GroupMail.GoldFluteSales.Value = True Then
       Set newRecipient = Item.Recipients.Add("Gold Flute Sales
Team")
       newRecipient.Resolve
     End If
     If GroupMail.BlueHornDesign.Value = True Then
       Set newRecipient = Item.Recipients.Add("Blue Horn Design
Team")
       newRecipient.Resolve
     End If
     If GroupMail.BlueHornSales.Value = True Then
       Set newRecipient = Item.Recipients.Add("Blue Horn Sales
Team")
       newRecipient.Resolve
     End If
     If GroupMail.Marketing.Value = True Then
       Set newRecipient = Item.Recipients.Add("Flute and Horn
Marketing")
       newRecipient.Resolve
     End If
     Cancel = True
  End If
End If
End Sub
```

This may seem like a lot of code, but it's actually just four lines repeated five times, once for each of the check boxes in the dialog box.

For each checkbox, we check to see if the user checked it. If the user did check it, we add the appropriate recipient to the mail message. We then resolve the name of recipient (have Outlook associate it with a real name) using the Resolve method of the Recipient object.

If you want to actually send the mail automatically, remove the line, "Cancel = True."

Note: The program will fail if it tries to send mail after resolving names that don't exist. To make it work, create a distribution list (File.New.Distribution List) in Outlook with names identical to the recipients you specified ("Gold Flute Sales Team," "Blue Horn Design Team," etc.) in the code written in this step. Add at least one address to each distribution list you create.

Hands-on Challenge #8

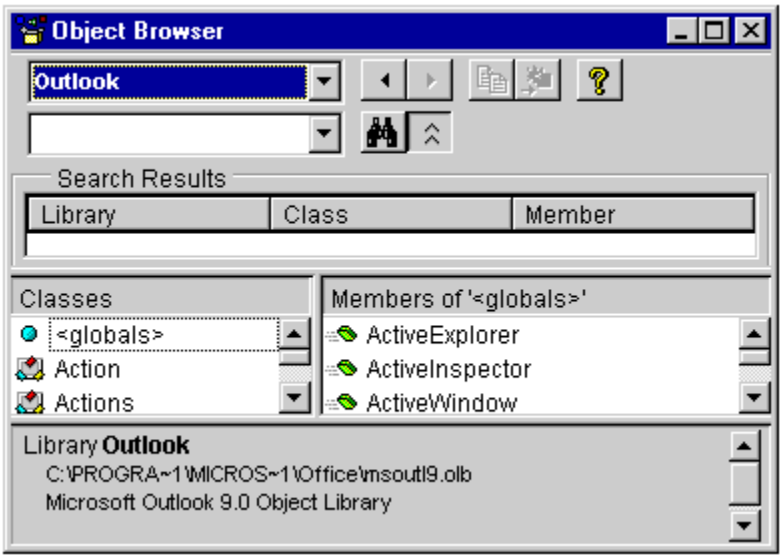
Make this lesson work for you! Think of how you could apply this program to your work. Change the code and the dialog box to work with five (or more) distribution lists that you often have to send important mail to.

Where to Go from Here

Now you have been introduced to the Outlook object model. You can now create custom Outlook solutions.

Of course, this guide only scratched the surface of all the powerful things the Outlook object model can do. Use the online Help to explore the many collections, objects, methods, and events at your disposal.

To use the online Help, from VBA, select Object Browser from View menu. You will see a dialog box that allows you to find all the tools available to you.



To see the Outlook specific tools, select Outlook from the top left dropdown list in the Object Browser. After Outlook is selected, click on the Help icon (the "?" graphic on the top right of the window) to view the online Help associated with the Outlook object model.

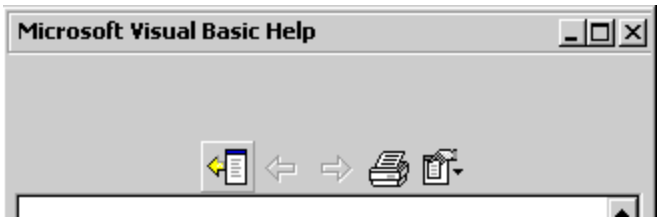
A window will show up on the right of the screen. The boxes you see in this window represent objects and collections of objects in the Outlook object model. (You can tell which is which by the key at the bottom of the window.)

Look at the top box labeled "Application." Every time you used ThisOutlookSession in your code, you've been using an Application object. Try clicking on the "Application" box. Here, you get all sorts of information about this object. There are also many links which can take you to related topics.

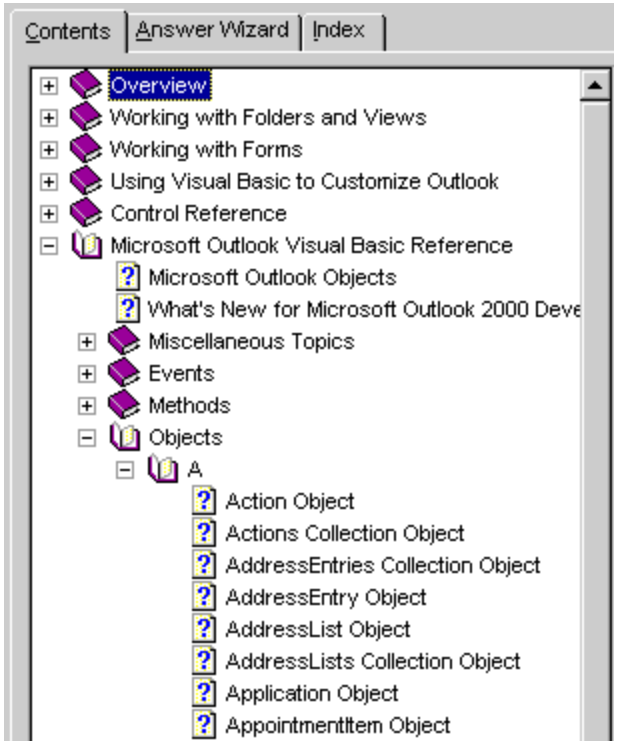
Try clicking on the "Methods" link in blue near the top of the window. The window that pops up shows you all the methods that are associated with the Application object. For instance, you'll see the CreateItem method, which we

used in the very first example in this guide. Double click on that method (or any method) to learn more about it.

You can also use the Contents in Help to get help on a specific area of the Outlook object model. From the help window, click on the Show icon (the one with the yellow arrow pointing left) to see an expanded view of the Help.



From here, you can explore all the different capabilities of the Outlook object model. For instance, let's say you wanted to work with appointments. Click on the Contents tab, and look for the Appointment object under Microsoft Outlook Visual Basic Reference.



All the information you need is here. And all the help files have hyperlinks in them so you can quickly navigate between related subjects, for instance to see what elements or methods a certain object has. Many of the help files also have examples that you can learn from and use in your own code.

Congratulations on becoming a real Outlook object model programmer! Now you can start using the Outlook object model to save your company time and money!

Appendix: Hands-on Challenge s Answers and Explanations

Hands-on Challenge #1

Try setting the body of the message to "This is where the content of the message goes."

To set the body, we follow the model of the line where we set the subject of the mail item:

```
Sub MyFirstMacro()
   Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
   NewMail.Subject = "(your name here) says: This really is easy! :)"
   NewMail.Body = "This is where the content of the message goes."
   NewMail.Display
End Sub
```

Hands-on Challenge #2

Open up the MyFirstMacro macro in the Visual Basic Editor and change the code to use a new procedure (i.e., MakingMail) to create a new piece of mail five times.

To create five pieces of mail, we need to call the procedure MakingMail from MyFirstMacro five times:

```
Sub MakingMail()
   Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
   NewMail.Subject = "Hey, this Sub/End Sub thing is easy too!"
   NewMail.Display
End Sub

Sub MyFirstMacro()
   MakingMail
   MakingMail
   MakingMail
   MakingMail
   MakingMail
End Sub
```

Hands-on Challenge #3

Look at the following make-believe object model code and determine which parts are objects, methods, and properties (there are three of each):

```
Set PetStore = ShoppingMall.GetStore(aPetStore)
PetStore.OpeningTime = 9 AM
Set Dog = PetStore.GetPet(aDog)
Dog.Breed = "Cocker Spaniel"
Dog.Color = "Blond"
Dog.WagTail
```

Objects	Methods	Properties
ShoppingMall	GetStore	OpeningTime
PetStore	GetPet	
Dog	WagTail	Color Breed

GetStore is a method, something the object ShoppingMall can do. GetPet is a method, something the object PetStore can do. WagTail is a method, something the object Dog can do. OpeningTime is a characteristic of the object ShoppingMall. Breed is a characteristic of the object Dog. Color is another characteristic of the object Dog.

Hands-on Challenge #4

To create a new macro that polls your co-workers for their breakfast choice, create a macro like the PollRestaurant macro and call it PollBreakfast:

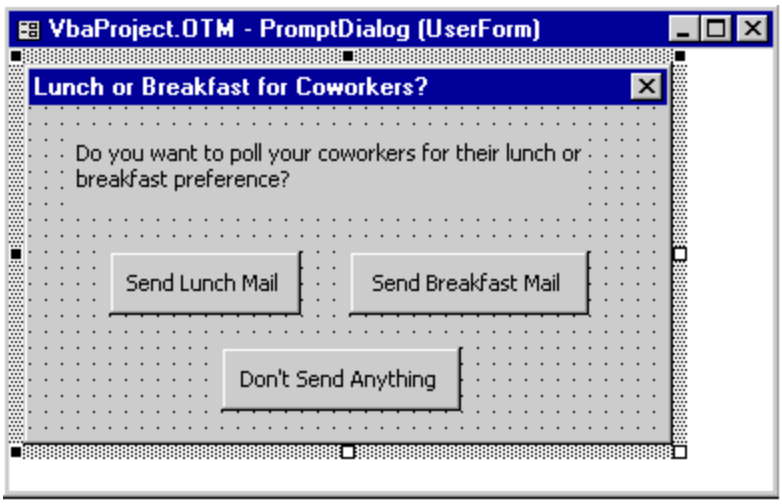
```
Sub PollBreakfast()
 Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
 NewMail.Subject = "Please open and make your vote for breakfast!"
 NewMail.Body = "Click one of the buttons to make your vote." &_
         "I'll be tallying first thing in the morning." &_
         "Breakfast will be in the break room at 9 AM."
 NewMail.VotingOptions = "Wake Up Land; Breakfast Blamos;" &_
                         "Eggs n'Stuff"
 Set receiverOfMyMail =
 NewMail.Recipients.Add("hungrypeople@mycompany.com")
 NewMail.Send
End Sub
```

Next, because you want this to run every time you quit Outlook, call the PollBreakfast macro on the **ThisOutlookSession** object's **Quit** event (in the ThisOutlookSession Module Window):

```
Private Sub Application_Quit()
   PollBreakfast
End Sub
```

Hands-on Challenge #5

First, we need to make some changes to the dialog box. Create a new button and make other adjustments to make the dialog box look something like this:



Double click on the "Send Breakfast Mail" button and make it so that you call your PollBreakfast module and hide the dialog box when the user clicks it (we assume the name of the button is Breakfast):

```
Private Sub Breakfast_Click()
      PollBreakfast
      PromptDialog.Hide
End Sub
```

Now you need to change ThisOutlookSession's Quit event (in the ThisOutlookSession Module Window) to show the PromptDialog dialog box when the user quite Outlook:

```
Private Sub Application_Quit()
   PromptDialog.Show
End Sub
```

Hands-on Challenge #6

Here is the code:

```
Dim numberOfPapers
Set paper1 = MyDesk.Papers.Add("The Foofle Report")
Set paper2 = MyDesk.Papers.Add("The Mooble Report")
currentNumber = MyDesk.Papers.Count
Set paper3 = MyDesk.Papers.Add("The Garble Report")
Set paper4 = MyDesk.Papers.Item(currentNumber)
Set paper5 = MyDesk.Papers.Item(1)
Set paper6 = MyDesk.Papers.Item(currentNumber + 1)
```

Here are the questions about the code with their answers.

1. What is the value of the variable, `currentNumber`?

   Answer: 2

   Explanation: When we set the value of `currentNumber`, there are only two papers, `paper1` and `paper2`, in the **MyDesk.Papers** collection.

2. What is the value of `paper3`?

   Answer: The Garble Report

3. What is the name of `paper4`?

   Answer: The Mooble Report

   Explanation: Because `currentNumber` equals 2, this sets `paper4` to the second paper in the collection, The Mooble Report.

4. What is the name of `paper5`?

   Answer: The Foofle Report

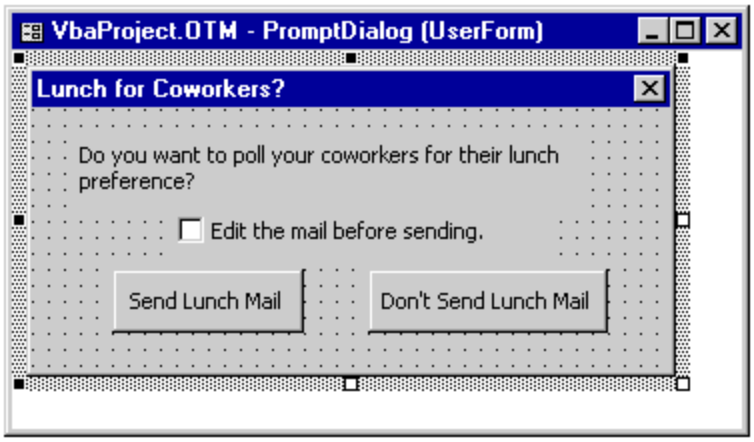
5. What is the name of `paper6`?

   The Garble Report

   Explanation: Because (`currentNumber` + 1) equals 3, this sets `paper6` to the third paper in the collection, The Garble Report.

Hands-on Challenge #7

First, we need to add a checkbox with the name "EditMail" to the **PromptDialog** form:



Next, we need to create an alternate PollRestaurant macro (called PollRestaurant2) that does not send the mail after creating it. In the same Module Window where PollRestuarant is, create PollRestaurant2. (Use copy and paste, add a "2" to the name of the procedure and remove the last line that sends the mail.)

```
Sub PollRestaurant2()
      Set NewMail = ThisOutlookSession.CreateItem(olMailItem)
      NewMail.Subject = "Please open and make your vote for lunch!"
      NewMail.Body = "Click one of the buttons above to make " &_
                  "your vote. I'll be tallying at 11 AM. " &_
                  "Lunch will be in the break room at noon."
      NewMail.VotingOptions = "Taco Temple; Burger Palace; " &_
                "Chicken Chums; Pasta Hut"
      Set receiverOfMyMail =
NewMail.Recipients.Add("hungrypeople@mycompany.com")
End Sub
```

Now, when the user clicks the Send Lunch Mail button, we need to call either PollRestaurant or PollRestaurant2 depending on whether **EditMail** is checked:

```
Private Sub Yes_Click()
   If PromptDialog.EditMail.Value = False Then
      PollRestaurant
   End If
   If PromptDialog.EditMail.Value = True Then
      PollRestaurant2
   End If
  PromptDialog.Hide
End Sub
```

Hands-on Challenge #8

There is no specific solution to Hands-on Challenge #8. You simply need to make adjustments to the code and to the dialog box to make the macro work for you.

◆ ◆ ◆ ◆