I’m Rob Burke, and I’ll be your host for the next hour. I work as an instructor and writer and all-around educational specialist here at ESRI. Also joining us is fellow instructor and ArcObjects™ specialist Jeremiah Lindemann. During the presentation you’re encouraged to ask questions and during those times when we can answer them, Jeremiah will be assisting. Welcome, Jeremiah, and thanks for joining us.

(Jeremiah) It’s great to be here. Thanks, Rob.
Seminar overview

- Topics include:
  - Understanding ArcObjects
  - Reading object model diagrams
  - Using programmer interfaces

- Format
  - Each topic followed by a question and answer period
  - Software demonstration

Let’s go ahead and get started. We’ve got three topics to get through today. Each topic will be followed by some questions and answers and a demonstration at the end. The first topic is "Understanding ArcObjects." There I’ll try to explain the basics of ArcObjects to give you an overview. After that, we’ll look at reading object model diagrams. There I’ll try to explain how to use the ArcObjects diagrams to help you write some code. In the end, we’ll talk about programmer interfaces—not user interfaces, but programmer interfaces. You’re not sure what they are? Stay tuned.
Understanding ArcObjects

The first topic is "Understanding ArcObjects."
The ArcGIS® applications, like ArcMap™ and ArcCatalog™, are built from ArcObjects. These objects are basic building blocks, so things like maps and layers and buttons and tools and the points and lines and polygons, and, in ArcCatalog, the tree view and the different things you can click on in the tree view. Each of those things are building blocks. They are created from ArcObjects.
ArcObjects and ArcGIS applications

- ArcMap and ArcCatalog are built with ArcObjects
- Each GIS part is a programmable object

ArcObjects are really programmable pieces that are assembled to build ArcMap and ArcCatalog.
ArcObjects and ArcGIS applications

- ArcMap and ArcCatalog are built with ArcObjects
  - Each GIS part is a programmable object
  - Users and programmers use same objects
  - Names may differ: programmer’s Map = user’s Data Frame

Users and programmers work with the same objects. You may know what some of these objects are already because you’ve worked with the ArcGIS Desktop applications. So you probably know point and line and polygon and maps and layers. You might call maps "data frames." The names might be different sometimes, but either way there are these objects that are the building blocks. Users work with them and so do programmers.
What are ArcObjects?

- Technically called classes
  - 2,700+ ArcObjects classes
  - Each class represents a basic GIS part

ArcObjects really aren’t objects at all. They are a set of classes. There are 2,700 of them altogether. Each one represents some basic building block of GIS.
What are ArcObjects?

- Technically called classes
  - 2,700+ ArcObjects classes
  - Each class represents a basic GIS part
- Coded in C++ by ESRI
  - Other programmers and users make objects out of them

Programmers at ESRI write C++ code to build these classes. So, a class like the Layer class, there’s code for it behind the scenes that’s written in C++ to get a layer to look the way it’s supposed to—to let it have a legend and color options and symbol options. Programmers write C++ code to make all that happen. Then, other programmers and users, like ourselves, we make objects out of these classes, and then go about setting properties and getting the objects to do different things.
Classes are stored in files—DLL files in your bin folder wherever you have ArcGIS installed. These are dynamic-link library files; they hold all the C++ code in a binary format. One file that is a key to all the others is called the esricore.olb. It’s an object library file that keeps track of all the classes. You don’t usually work with these files. You don’t ever go to these folders and work with these files. However, if you do any future programming work, you might reference these and take advantage of the code that’s already been written.
What are objects then?

- You see them (sometimes) and set their properties
- Stored in active memory
- Can be saved in a map document

Objects, then, are different than classes. Objects come to life. You can see objects on your screen sometimes. When you start ArcMap, an ArcMap application window is created. It’s an object. It only lives in active memory, and it displays on your computer screen. Layer objects can be created. You can make as many of these layer objects as you want and then you go about setting their properties to get them to look the way you want. You set their color and their symbols.

Objects only live in memory, so those layers are stored in active memory with the current property settings. If you were to quit ArcMap, those objects would go away. Before quitting, if you saved your work into a map document, the MXD file would contain the parameters that you set—the properties that you set—so that next time, when you start ArcMap up again, it could recreate those same objects and they would look the same way as they did before.
Classes vs. objects

- **Classes**: live as code in files
  - Its code controls how its objects work
- **Objects**: live in memory
  - You create objects from classes
  - You create many Layer objects from the Layer class

Classes are code that live in a DLL file. Here we have the Layer class. From the Layer class, users create Layer objects. Programmers also create Layer objects from the Layer class. Programmers do it with code; users do it by clicking the Add Data button. When you go about changing a layer’s colors, symbols, and properties, you’re able to do that because the code in the Layer class allows you to do that.
One of the ESRI classes is called the ContentsView class, and you might know the objects created from ContentsView: they’re the Display and the Source tabs in your table of contents window.
I used Visual Basic to make my own ContentsView class; I called it Rob’sView. From that class, I’ve created a Catalog object. Whenever ArcMap starts up, my class has been registered with ArcMap, so it knows to create these Catalog tab objects whenever it starts up.

Let’s take a look at this tab to see how I got it to work [opens ArcMap]. If I click on the Catalog tab, I see that I’ve got an ArcCatalog tree view. Here I can navigate through files and folders, just as you would when you use ArcCatalog, and then I can find a feature class and drag it onto my map. If I look at the Display tab, I’ll see that, oh yeah, there’s the new layer added to the map.
You can use a variety of programming languages. As I mentioned earlier, I used Visual Basic to create that ContentsView class. The ESRI programmers work in C++. Maybe one of the more popular languages is VBA, or Visual Basic for Applications. I say that because it comes with the ArcGIS applications—it’s built into ArcMap and ArcCatalog. You don’t have to make an extra purchase to get VBA; you’ve already got it if you have ArcEditor™, ArcInfo®, or ArcView®.
Well, to review this bit so far, we’ve talked about how ArcObjects are really a whole bunch of classes stored in DLL files, and how users and programmers alike make use of those classes by making objects out of them. Objects come to life in memory, in active memory on your computer. When you’re done with one of the applications, all the ArcObjects are removed from memory. Classes are always there; they’re always out there stored in the DLL files.

Well, Jeremiah’s got a whole bunch of questions listed here. Let’s try to answer a few of them.

(Jeremiah) Great. Let’s take some questions that you guys have been sending in through the Ask the Presenter button on your display.

Ken from Klamath Falls, Oregon asked, “If I write something using ArcObjects from 8.1, will it work in 8.3?”

And the answer is yes. This is part of what’s called the COM contract. All ArcObjects are built upon the specified structure where they’re guaranteed not to change, and they’ll work from version to version. Some of you may have heard that we’ve been working in ArcGIS 9 and we’ll
be releasing it this year. And, anything that we’re writing in ArcObjects 8.3 will also be working within ArcGIS 9, as well. That’s nice to know.

Ken from Dallas asks, “Can you build ArcObjects that run from Internet Explorer?”

This is a little bit of an advanced question, and, no, we cannot use ArcObjects through Internet Explorer at ArcGIS 8.3. Something else in ArcGIS 9 that will be coming out, there’s going to be a new product called ArcGIS Server that will allow us to do this.

Sally from Jacksonville asks, “Do I have to buy ArcObjects?”

The answer to that question is no, you don’t have to buy ArcObjects specifically. When you have ArcGIS Desktop, you automatically get access to ArcObjects. Just as Rob was mentioning, you get access to VBA, where you’ll be accessing the objects. The objects are already there. They’re included with the application. Okay, I’ll turn it back over to Rob now.

(Rob) Alright, thanks, Jeremiah. Let’s go ahead and move on to the second topic.
Here we’re going to look at reading object model diagrams.
Maps are diagrams

- Say you are going to a new city
  - Get a map to learn the city roads and landmarks
  - The map looks complex at first
  - Use the legend to understand the map symbols

Object model diagrams are almost like road maps of the ArcGIS system. Similar to the way you’d use a map to learn a new city, you use the object model diagrams to learn how the ArcObjects are organized. If you go to a new city, like Albuquerque, you might get a map of that city and try to familiarize yourself with the roads and the landmarks and try to find your way around the town before you actually go there. You use the map to do that. You can understand the map and its symbols because there’s a legend there, and it tells you what all the colors mean—the red are interstate highways, the black are regular highways, and the gray ones are streets.
When reading the ArcGIS object model diagrams, they look a little bit complex at first too, but they are like road maps for the different groups of ArcObjects. Earlier I mentioned how there are 2,700 classes; there are about 30 diagrams to show those classes. Each of the diagrams is organized by topic. So, for example, the diagram you see here is the ArcMap object model diagram, and it shows all the classes related to creating the ArcMap display, ArcMap window, ArcMap application. There’s another set of objects called the Geometry objects. They have their own diagram. The Display diagram contains classes like renderers, for making layer legends, and color models, to change colors and symbology for points and lines and polygons.
Object model diagrams

- Like maps for the different groups of ArcObjects
  - ArcMap, Display, Geometry, etc.
- Use the legend to understand the diagrams

Each of the diagrams has a legend full of symbols to help you understand how to read the diagram. In the next few slides, I’m going to explain what all those symbols mean in the legend so that you can read the diagrams and learn to write some code.
The diagrams use, or are drawn with, the Unified Modeling Language. You see a lot of squares and lines connecting them and different symbology there. We tried to follow a standard when creating these diagrams, called UML. It’s an industry standard for diagramming object relationships.
### 12 basic UML symbols

- **Relationships**
  - △  ◆  ⋯→  ⋯

- **Classes**
  - ![Class Symbols](image)

- **Properties and methods**
  - ![Property and Method](image)

UML is made up of many symbols, but you only need to know about 12 of them to really navigate your way around the diagrams and start to write some code. I will define how to use these symbols in the next few slides.
The one-liner symbol is called the association. These two classes, or, if you create objects out of these two classes, the objects are associated with each other. The way we’d read this is that a chicken has an associated nest. The relationship here is one-to-one. There’s just one line and no other symbols, so one chicken has one nest. If you saw one of these objects all by itself, you really wouldn’t think anything odd of it, but they usually go together in some way.
The association can be more than just a one-to-one relationship; it can be a one-to-many relationship. The star here signifies a many, or multiplicity, relationship. The class the star is closest to, that’s the Many object. So here we would read this relationship: a farm has many chickens.
The multiplicity relationship can have an actual number. Here a chicken has two wings. The number is closest to the object that has the multiplicity.
The "composed of" symbol is shown with a solid diamond. "Composed of" means that the object with the diamond is the composed-of object. So we’d read, this chicken is composed of two wings. Composed of means that these two objects go together all the time; you wouldn’t see one without the other. If you were moving the Chicken object, its composed-of objects would go with it. If you deleted this object, the composed-of objects would get deleted also.
The dashed line and the arrow indicate a "creates" relationship. Here, one type of an object creates this other one: chickens create eggs.
The last relationship, here, is shown with a triangle. You read this one from the bottom up. So, starting down low and going up towards the class that the arrow or the triangle is pointing up towards, you would say that a chicken is a type of bird. When you see this symbol, you should think inheritance. There’s inheritance going on here. Birds have some properties and methods that are inherited by chicken. Sure, all birds have wings and feathers and a beak. Chickens are a type of bird, so they have those properties too. Inheritance does not go down here to these objects. This is not an inheritance symbol or "type of" symbol. So, inheritance only happens when you see the triangle.
The Bird class is called an abstract class. It’s shown as a two-dimensional box shaded in. Abstract classes mean that you cannot make any of these objects. Birds do not exist. You might be saying, "Wait a minute, Rob, I can see a bird right now outside my window." And I would say, "No. No, you do not. You see a specific type of bird."
You see a robin or a sparrow or a cardinal, but, in reality, there are no birds. There are types of birds. The abstract class is really just there for generalization and inheritance purposes, to hold those common characteristics that are inherited by any subclasses. All birds have wings, feathers, and beaks.
CoClasses are shown on the diagrams in a three-dimensional box, and they’re shaded in. When you see a CoClass, this means that you can create objects from these classes. Any time you see a CoClass, it means that you can write two lines of code to make one of these objects. These are your starting points. If you’re not sure where to start writing code, you can locate a CoClass on any diagram and, in two lines of code, you can make an object out of it.
I’ll show you the code, here, in just a minute.

Classes are shown on the diagram also as 3D, but they’re not shaded. You cannot make objects out of these classes. You cannot make eggs or wings. Other objects have to do the creation for you. You can see it on the diagram here with the "creates" relationship symbol. Chickens create eggs. If you want an Egg object, you have to go to the chicken and have it make the egg for you. If you don’t have a chicken, it’s a CoClass, so you could make one with two lines of code and then get it to make the egg.
All classes have properties on them shown with the barbell symbol. Here, the Chicken class has three properties: age, color, and name.
Some of the properties have a left-sided barbell. The left-sided barbell means you can only get this value, but you can’t change it to something else. So here, for chicken, you could get a reference to its wing, but you couldn’t give it a different wing. You couldn’t take a wing off an eagle and put it onto a chicken.
Some of the properties are called write-only properties. They’re shown with a right-sided barbell. You can edit this value, but you can never find out what it is. So you could change the chicken’s password, but you can never know what the current password setting is.
To the right of each property are a colon and then a type. This is the value that the property holds. Age holds an integer value. Integers are numbers like 1, 2, 3, 4, and 5. Some of the properties hold the basic data types like numbers and strings and dates and Boolean values. Some properties will hold objects or references to objects. So here, the wing property would give you a reference to a Wing object.
Classes also have methods. Methods are shown with an arrow symbol. Methods are like the actions that an object can perform.
Methods can return values

- Some methods return a value, some don’t
- Return values have a type
  - Method: Type

Some methods can return a value. Some methods don’t. Fly does not return any value. It carries out its operation and that’s it. LayEgg, on the other hand, carries out its operation and returns an Egg object to you. This is how you carry out that creates relationship. This is how chicken creates eggs with the LayEgg method.
Well, let’s start to write some code here. We have the Chicken class to start off with. It’s 3D. It’s shaded in. It’s one of those CoClasses. That means you can create an object out of this class with two lines of code. To create an object, declare a variable. For the type, after the As keyword, use the class name. To create an object, then, set that variable using the new keyword and the class name.
Now you have an object in memory and, more importantly, a variable to refer to that object.
One class: many objects

- Declare more variables
- Create more objects
- Each object has an object variable

Dim c As Chicken
Set c = New Chicken

Dim c2 As Chicken
Set c2 = New Chicken

Dim c3 As Chicken
Set c3 = New Chicken

With CoClasses, you can make as many objects as you want out of them. So, the whole time there’s only one Chicken CoClass—it’s code is stored off in a DLL file somewhere—but from that Chicken CoClass, you can make as many chickens as you want. Just declare variables and set them using the New keyword.
Say you want to change a property, like the color property, or say you want to get the color property. When you look on the object model diagram, you always look to the right of the property to find out what its data type is. Here, color is a string.
If you want to get the color property, declare a variable as a string, set that variable using the equals sign, and then the Object.Property syntax. Your variable, here, refers to the object. Here’s its color property. This returns the value that’s stored in color and puts it in the variable `x`. 
If you want to use that value, you might print it up in a message box to find out what color the chicken is.
To change the chicken’s color, you use the same syntax: Object.Property and the equals sign. Here I enter a string that’s quoted. I use a string because I know Color expects me to set its value using a string data type.
To run methods, the syntax is very similar: Object.Method.
Some methods have an arguments list with parentheses to the right of the method name.
If the arguments list has arguments in it, the arguments will also have a data type. Here, Fly has a speed argument and, in order to get Fly to run, you’re going to have to enter a string in the line of code, and that is going to have to be some text string describing the speed. So here I have Fly and then a text string describing what speed I would like the chicken to fly at. If there were more arguments, I’d add a comma after the first argument and then that second argument after that.
Some methods return a value. LayEgg returns an egg. According to this creates relationship, this is how you will get to eggs. Egg is a class. That means you can’t create an egg yourself—you’ve got to get the other object to do it for you, and the way to do it is with the LayEgg method on Chicken. If you want an egg, use the egg data type to declare a variable, or here it’s an object type or class name.
Then set that variable using the LayEgg method on Chicken. After that, you’ve got a variable, now, to refer to your Egg object.
Code to run a method to return a value

- Declare a variable as the return type
- Set the variable using the Object.Method syntax
- Use the new object's properties and methods

```vba
Dim e As Egg
Set e = c.LayEgg
```

If you want to do something with that variable—that object variable—like check the color, you might print that up in a message box.
Any time you want to get to a neighboring object, there’s always a property or a method on your current object to get you to the neighbor. Just earlier, there, we looked at how LayEgg gets eggs. Well, if I want to get to the chicken’s nest, there’s also a property there to get to the nest.
So, to get to a neighboring object, you need to write two lines of code: declare a variable as that neighboring object…
Code to get an associated object

- Get any neighbor in two lines of code
- A property on the one returns the other
  - Chicken’s Nest property returns associated nest
  - Declare an object variable
  - Set it with Object.Property

```vba
Dim n As Nest
Set n = c.Nest
```

…and then set that variable.

Now I have two distinct variables: one pointed to the chicken, and the other pointed to the chicken’s nest. I may need that nest variable because the chicken could leave, and I may need to check status from time-to-time to see if the chicken is in or out. The nest may also have a Count property, so I can get a count of how many eggs are in the nest.
Alright, here in this section, we’ve talked about object model diagrams. I’ve tried to show you how there are about 12 different symbols, and once you know those symbols you’ll be able to start writing some code. There are a couple of steps that you always go through when writing ArcObjects code. First, you need to find a starting point. You can start with a CoClass because you know you can make an object out of a CoClass by writing two lines of code: declaring a variable and setting it with a New keyword. Once you have an object, well, then you can start using properties and methods on that object. Sometimes you need to get to neighboring objects or create the neighbors. You can do that in two lines of code by declaring a variable for the neighbor and setting it with a property or method on the object you currently have. Once you have that neighboring object, you can run its properties and methods.

Alright, Jeremiah’s got a couple of questions here. We’ll go ahead and answer them.

(Jeremiah) Great. Thanks, Rob. Okay, Matt from Denmark asked, “Can you create your own classes in VBA?”

Yes, you can create your own classes in VBA. But, remember, as Rob pointed out there are nearly 3,000 ArcObjects classes that are already available. So most people that are using
ArcObjects really have no need of creating their own classes since they are already supplied for you. This is an advanced thing that can be done and, usually, when it is done, it’s usually done in a standalone programming language, such as Visual Basic.

Okay. Sarah from Fresno asked, “If I have VBA, how can I create an EXE—an executable—or a DLL format package?”

Well, an EXE is just a style—an application or DLL is just another component that can plug into another application. With VBA—Visual Basic for Applications—we cannot create EXEs or DLLs. You have to go into a standalone programming language, such as standalone Visual Basic. So, the answer is you cannot do it in VBA, but you can package up ArcObjects into these formats in programs other than VBA.

Similar question: Margo from Hood River asks, “Can I write my own DLLs that distribute to users that don’t have a copy of ArcGIS installed?”

So, if I do create these other formats—if I want to supply them to other people, they also have to have ArcGIS installed on their machine if they want to use ArcObjects. So anywhere you want to use ArcObjects, that machine also has to have ArcGIS Desktop installed. This’ll be changing a little bit in ArcGIS 9. There’ll be new products that will enable you to deploy ArcObjects without a license of ArcGIS.

Okay. And I guess we’ll turn it back over to Rob right now.

(Rob) Alright. Thanks a lot, Jeremiah.
We’ll go on to the next section, here, "Using Programmer Interfaces."
All of the ArcObjects classes are created using a standard called COM, or Component Object Model. COM is really just a set of rules that the ESRI programmers follow to create their classes. By following these rules, the classes become programming language-independent. The code for the ArcObjects classes is written in C++, but you can access those classes and run their code from all kinds of other programming languages. We’ve been using VBA or VB, but you can use a variety of other languages too. Also, it allows the COM classes to work within other applications. If each application follows the COM standard and all the classes between several applications follow the standard, then the objects between them can kind of be mixed and matched and work together. And here’s an example of that.
I’ve been presenting to you using PowerPoint and, in my presentation here on this slide, I’ve added a data frame. You might be looking there and thinking, well, that sure looks like you just imported a JPEG or a BMP image. Well, no, this is a data frame—this is an ArcMap data frame. It comes from the ESRI map control, and it’s got some special, built-in functionality where I can zoom in on these layers—I’ve kind of customized the layers, so when I zoom in on the States layer, the labels know when to come on based on the scale. As I zoom in a little bit further, the layer knows when to turn off and a new layer comes on: the Counties layer. As I zoom in on the Counties layer, its labels know when to turn on based on scale. I’ve programmed a couple of buttons here. I’m working in PowerPoint’s VBA. PowerPoint also has VBA built in to it.

So, I made a button here called Zoom All, and I wrote ArcObjects code to get it to zoom to the extent of all my layers. I created another button that goes into the layer and selects one of the states and zooms to that state based on user input, here. So I’ll type in a state name. I’m from Wisconsin, here, so I’ll zoom to Wisconsin and the code knows to then zoom in to that state. So here I’ve got ArcObjects working inside of PowerPoint. The reason that can happen is because the PowerPoint classes and the ArcObjects classes are created according to the COM standard, so they work together and I can mix and match them between applications as I need them.
The way this all works is that the COM classes have something special added to them called an interface, or a programmer interface. This is not a user interface, but it’s actually built into each class. When classes have interfaces, they can communicate with each other even though the languages are different. So for example, the ESRI programmers write the ArcObjects classes using C++ code. Well, you could program with those objects using other code—other programming languages. So here, the language really doesn’t matter if the classes were created according to the COM standard. An interface is quite detailed, and I just don’t have enough time to go into them here, but you might think of a programmer interface as sort of a translator that allows code in one language to communicate with code in another language. There’s a lot more to it than that, but that’s about the most simplified version.

I have to save my map work, here in my PowerPoint presentation, and we’ll continue on.
Programmer interfaces

- Not a user interface
- Group properties and methods
- Lollipop symbol

On the diagram, interfaces are shown with the lollipop symbol. All the ESRI class interfaces start with the letter "I." So if dog happened to be an ESRI class—it’s not, but if it was, it would have the IDog interface.

We were saying earlier in the lecture that a class has properties and methods. Well, just put interface in between there, and then the way we talk about it is a class has interfaces and then the interfaces have properties and methods. Any time you want to work with the ArcObjects classes, you must work through their interfaces.
Now, what that means to you as a VB or VBA programmer is, when you’re declaring your variables, you declare them directly to one of the interfaces instead of the class. So really, the only thing different here about programming with interfaces is in declaring your variables, choosing one of those interfaces to declare to.
After that, when you create new objects, it’s the same code to set a variable, use the New keyword to make a new object, and put the class name on the end.
Once the object is created, you have access to properties and methods on the interface that the variable has been declared to.
All COM classes have many interfaces. That means when you declare variables you have to make a decision: which interface should you use? You make that decision really based on the property or method you need.
Programming with multiple interfaces
- Classes can have multiple interfaces
- Declare variables to the interface you need
  - Need a blue dog: Use IDog interface and Color property

Dim d As IDog
Set d = New Dog
d.Color = "Blue"

If you need to change the Color property—Color is on the IDog interface—you would declare your variable as IDog.
Programming with multiple interfaces

- Classes can have multiple interfaces
- Declare variables to the interface you need
  - Need a blue dog: Use IDog interface and Color property
  - Need a dog named Jake? Use IPet and Name

```vba
Dim d As IPet
Set d = New Dog
d.Name = "Jake"
```

If you needed to change the Name property, Name is on the IPet interface, you would declare your variable as IPet, create a new object, and then go about setting its Name property. So, when programming with the ArcObjects classes, you find the property or method you need, then declare a variable as the interface that contains that property or method.
As a naming convention, we use the prefix “p” before any of these variable names that point to interfaces. “P” stands for pointer, and the way we talk about these variables is that they point. So here, I’ve declared a variable, pPet, as the IPet interface, and the way I talk about it is that this variable, pPet, points to the IPet interface.

When naming the variable, we use the prefix “p,” and then the name of the interface without the “I.” By doing that, when other people look at your code, well they can just look at this variable, if they’re maybe down deep in your code and haven’t seen where you’ve declared that variable, they can look at it and say, "Oh, you’ve used the naming convention. This must be pointing to the IPet interface."
Well, let’s look at an example here, and create some objects and work with their properties and methods. Alright, so I want to create a dog and set its color.
How does it work?

'Create a brown dog
Dim pDog As IDog
Set pDog = New Dog

The Color property is on the IDog interface.
How does it work?

'Create a brown dog
Dim pDog As IDog
Set pDog = New Dog
pDog.Color = "Brown"

That’s the way I declare my variable, and then I can set the Color property.
'Create a brown dog
Dim pDog As IDog
Set pDog = New Dog
pDog.Color = "Brown"

'Create a dog named Benny

Say I want to change the name of a new dog.
How does it work?

'Create a brown dog
Dim pDog As IDog
Set pDog = New Dog
pDog.Color = "Brown"

'Create a dog named Benny
Dim pPet As IPet
Set pPet = New Dog

Well, Name is on the IPet interface, so I declare my variable as IPet, create the new dog,
How does it work?

'Create a brown dog
Dim pDog As IDog
Set pDog = New Dog
pDog.Color = "Brown"

'Create a dog named Benny
Dim pPet As IPet
Set pPet = New Dog
pPet.Name = "Benny"

and then set its Name property.
In the end, here, I end up with two Dog objects.
Review: Programming with interfaces

- You end up with two dog objects
- Each dog has a unique variable

There are two different variables referencing each of the unique dogs.
Review: Programming with interfaces

- You end up with two dog objects
- Each dog has a unique variable
- Each variable points to a different interface

Each variable points to a different interface because I needed to get to properties and methods on the different interfaces.
Switching interfaces on one object

'How about creating a brown dog named Bart?

Well, now, how about working with one dog? Let’s make one and then set its properties and methods, but using several of the interfaces.
Switching interfaces on one object

'How about creating a brown dog named Bart?

'Create a dog
Dim pDog As IDog
Set pDog = New Dog

Here I want to create a dog and change its color and its name.
Switching interfaces on one object

'How about creating a brown dog named Bart?

'Create a dog and Set Color
Dim pDog As IDog
Set pDog = New Dog
pDog.Color = "Brown"

So, I make the new dog—I use the IDog interface so I can get to the Color property. Now, I want to work with this same dog.
Switching interfaces on one object

'How about creating a brown dog named Bart?

'Create a dog and Set Color
Dim pDog As IDog
Set pDog = New Dog
pDog.Color = "Brown"

'Set Name
Dim pPet As IPet

I do want to change the Name property so I need a variable pointing to IPet. But then, instead of setting this new variable using the New keyword, I don’t want to make a new dog…
Switching interfaces on one object

'How about creating a brown dog named Bart?

'Create a dog and Set Color
Dim pDog As IDog
Set pDog = New Dog
pDog.Color = "Brown"

'Set Name
Dim pPet As IPet

'Don’t make a second dog
'Just set pPet equal to pDog
Set pPet = pDog

…so I’ll just set the variable equal to the variable I already have.
Switching interfaces on one object

'How about creating a brown dog named Bart?

'Create a dog and Set Color
Dim pDog As IDog
Set pDog = New Dog
pDog.Color = "Brown"

'Set Name
Dim pPet As IPet

'Don’t make a second dog
'Just set pPet equal to pDog
Set pPet = pDog

pPet.Name = "Bart"

After that, I can use properties and methods on the interface that this variable points to. The other variable is still there; it still points to its original interface, and I could still use that variable too.
Review: Switching interfaces on one object

- You have one dog object
  - Two variables to refer to that dog
  - The variables point to different interfaces
- This is called QueryInterface

What I end up with here is one Dog object, but two variables that reference that same dog. The two variables each point to different interfaces. I needed to get to properties and methods on those different interfaces, but for one object. This technique of working with one object and setting up many variables to point to different interfaces of one object is called QueryInterface, or QI for short.
Well, we’ll take a little break here, and there are a lot of questions in the queue, and I’ll have Jeremiah try to answer a couple of them.

(Jeremiah) Okay. We’ll get a few of these answered here for you. Okay.

Nick from Minnesota asked, “How do you integrate ArcObjects code into other applications such as PowerPoint? Do you place code into a specific location on the hard drive or does PowerPoint load the ArcObjects code into it?”

Well, you could really do this a couple of ways. You could work with a standalone application that calls objects from many different places. But probably the easiest is what Rob was showing; he placed the ArcObjects code directly inside PowerPoint. As we mentioned, when we have ArcGIS, we develop within Visual Basic for Applications. That’s an editing environment that automatically comes with ArcMap and ArcCatalog. Well, PowerPoint also has Visual Basic for Applications, which we can open up and start working with ArcObjects as well. So that’d be the easiest way.
Tom from Burbank has a similar question. “Just to clarify, if I wanted to nest a map in PowerPoint, like the demo, I still have to have ArcGIS loaded, right?”

And that is correct. We would also need to have ArcGIS loaded on our machine; that way, it knows where to get the ArcObjects from.

Lisa asked, “Remind me, what is a CoClass?”

Well, there are different types of classes that we can work with. We have abstract classes, CoClass, and regular classes that are sometimes referred to as instantiable classes. Most of this lecture we’re working with CoClasses. Now, a CoClass is a special type of class that can be created brand-new. Some objects we cannot create brand-new, and we have to have different types of class representation for those. For instance, I could go in and I could create a data frame programmatically, brand-new. That would be an example of a CoClass. There are other types of classes that I can’t create brand-new; for instance, a row in a table. Although I can create it brand-new, I have to get it from another object. So that’s a special type of class. I have to get that row from a table, so a table creates another object. So a CoClass is just a special type of class that we’re working with.

And Rob from Wyoming asked, “Where can I find the model diagrams for ArcObjects?”

Rob is actually going to go over that in the presentation, here in a little bit, and find out specifically where we can get that. But when we install ArcGIS Desktop, we get the option to install the developer kit. If we install the ArcGIS Developer Kit, the object model diagrams are all stored in PDF format automatically on our machine, in the same location where ArcGIS is stored. And, again, we’ll go over that in just a moment. We’ll turn it back over to Rob now.

(Rob) Alright. There was a question, "What if you write code at 8.1 and you want to use that same code at 8.3 or at 9, will that 8.1 code be valid?"

And one of the answers to that is yes. And it has to do with interfaces. If ESRI needs to change a class or change a property or method on an interface, one of the rules behind COM is that interfaces won’t change over time. So once we’ve released a version of the software, the interfaces on that version will not change, ever. If we need to make a change, we’ll add another
interface and carry out the changes on that new interface. If you decide when you get the new version of the software, that you want to take advantage of the new interface, you could move over to the new interface, but the idea is that your old code will not break when you load new versions of the software.

Alright. We’ll go ahead and continue with a little bit of a demonstration here.
We’ve been working with dogs and farm animals, and now it’s time to work with real ArcObjects classes. In my scenario, here, we’re going to work with a layer and change its name.

I bring up my ArcMap, here—earlier I added a layer, a Rivers layer. When you add a layer to a map, the name of the layer takes on the name of the feature class. So, here, my feature class had some odd coded name, and I want to change that name. Well, you might be saying, "Wait a minute, Rob, I could just right-click on that layer and change its name; that’d be easy enough for me to do." Well, sure, the user could make changes, but on the programming side, we can write code to carry out those changes for the user and make it a little bit easier.

Alright. So here we’re just going to work with the Name property on Layer. Name is a good example, because I’m going to have to do the five main things that you would normally do while programming with ArcObjects. The first thing you would do is take a look at the diagrams. Look at those diagrams and figure out how you’re going to write your code, and try to learn which classes you’re going to need. When you find out which class you’re going to need, a lot of times, on that class, there’ll be a lot of properties and methods scattered over several different interfaces. So you may have to use the QueryInterface technique to switch interfaces.
A lot of times you have to go to one neighbor to the next. Yeah, because you may start with one object, but the object you need is maybe two or three or four objects away, and you may have to move from one to the next until you get the one you need. Once you have that object, then you can get to its properties and methods and run them.
When you start ArcMap, several objects get created in memory automatically. If you start a new session of ArcMap, the Application object gets created. The Application object comes from the Application class and it refers to the ArcMap window—like the status bar and the collection of toolbars and everything that makes up the ArcMap window.

Also, an MxDocument object gets created. This is a map document. If you start a new session of ArcMap and you haven’t really opened an MXD file, there’s one there—there’s an MxDocument already there; it’s called Untitled. So these two objects are always there if you have ArcMap going. If you look at the relationships in the diagram, an application is composed of an MxDocument. There’s no multiplicity here, so it’s a one-to-one relationship. That means, whenever you open ArcMap, you always have a map document open, too. Map documents are composed of many maps. Here we have the one-to-many relationship, shown with the star. And then maps are composed of layers. Again, we have the star and the composed of relationship. Remember, maps are called data frames by the user. So here we have a world data frame—a world map—and then a USA map. Each map has multiple layers—the world only has one layer to it.
You might be thinking, "Well, geez, I know these relationships, since I’ve been working with ArcMap for a while. Of course I know that a map has layers and a document has many maps in it." Well, that’s good. As a user, you may know a lot of the relationships that are shown on the diagrams. The main purpose for the diagrams is for you to learn about how the objects are organized and, mostly, when you start to work with some new objects, how those new objects are organized and what their relationships are and what other objects they are connected to.
Application and MxDocument are two places where you can start your programming work. I say that because they are always available in VBA programming inside of ArcMap. Remember, if you find a CoClass, you can always start with a CoClass, too.

Let’s take a look at these two on the real ArcMap diagram.

To open the diagrams, you’d go to the ArcObjects Developer Help page by going to your Start button and wherever your ArcGIS is installed. The ArcObjects Developer Help is one of the choices there. On the bottom is an object model diagrams link. If you click that, you get a listing of all the possible object model diagrams. We want to go to the ArcMap diagram because we want to work with the application, the map document.

So, I’ll click on the ArcMap diagram to bring up Adobe Acrobat Reader. The diagrams are stored as PDF files. They’re somewhere in your installation folder whenever you install ArcGIS. If you’ve installed the developer kit, you have PDF files for each of these object model diagrams. They are poster size; this one is 42 by 34. You could print them out if you have a plotter large enough. If you like, you could go to the ESRI Store at the ESRI Web site, and you can always buy some of these diagrams. You could get the whole set already printed out.
Alright, on the diagram here, as I go to the top of the diagram and work my way down this ArcMap diagram, the first class that I come to is the Application class. It’s way up at the top; it’s the highest one. Let’s zoom in on Application to see what it looks like. As I zoom in on this class, I see that it is a class. It’s three-dimensional; it’s not shaded, so I can’t make an Application object here. I don’t need to because it’s already been created. When I start ArcMap, it’s always there and always available.

There are a lot of interfaces available. Each interface has properties and methods to it. If we look to the right here, there’s a composed of relationship, and as I pan here, I see that an Application is composed of one MxDocument. An MxDocument has its unique interfaces and each interface has its unique properties and methods.
Start with MxDocument

- MxDocument has a preset variable
  - Called ThisDocument
  - Points to IMxDocument

MxDocument has a preset variable, ready to use, already assigned for it. It’s almost as if someone declared and set this variable for you. It’s ready to go, ready to use.

We’ll go ahead and jump over to ArcMap and bring up the Visual Basic Editor and try out this code. Alright, I’ve got a test subroutine here that I typed in before the session and I’m going to use a message box. And here’s that ThisDocument variable. Notice that I didn’t have to declare or set this variable; it’s just ready to go, ready to use. When I type in the doc, I see a list of properties and methods.

What properties and methods are these?
Well, the ThisDocument variable, as with all ArcObjects variables, must point to some interface. ThisDocument points to the IDocument interface.

If we look at the object model diagram [goes to PDF] for MxDocument—let me scroll down here and find that IDocument interface, and I’ll zoom in a little bit so we can read this—the variable there, ThisDocument, points to IDocument. That means, through that variable we have access to these seven properties. They’re read-only properties so we can get their values.

Back to my code here [in Microsoft Visual Basic]. I’m getting ready to type in some code. As I use the code completion, here, I see that Title is one of the properties available. I’m going to select that and we’ll go ahead and run this line of code to display the map document’s title. And here it comes back and tells me the title is USA.mxd. You might use this code in some kind of an If Then statement. Maybe you want different sets of code to run depending upon the map document that’s open. Your If Then statement could get the title of the map document; if it’s one map document, one set of code could run; if it’s another kind of a map document, you might run another set of code.
Start with Application

- The Application object also has a preset variable
  - Called Application
  - Points to the IApplication interface
- Preset variables are only available for these known objects

ThisDocument is a preset variable. A second preset variable is also available called Application. Application points to the IApplication interface for the ArcMap application. These are the only two preset variables available in VBA, so you won’t see them really noted in any way on the object model diagrams. They’re the only ones available because these are the only known objects. When you start ArcMap, you always have these two objects available and ready to go and then you can add maps and layers and other things to the application or the map document to create other objects.
Alright. Remember, looking at the diagram, the IDocument properties were read-only [goes to object model diagram PDF] and there’s only about seven of them. Remember, we want to get to a layer and change the layer name. So let’s scroll up here. We’re going to need to look for another interface because IDocument had no way for us to get to a layer. Alright. Here I’m at the IMxDocument interface. It has a lot of properties and a lot of methods. One of them will get me to the active map. It’s called the FocusMap property. This gives you the active data frame. Down a little bit is the selected layer property. This returns the ILayer interface of whatever layer happens to be selected in the table of contents. We need this property; it’s on this interface, so we need this interface too.
Switch interfaces to IMxDocument

- IDocument's properties are read only

We’re going to have to switch interfaces. ThisDocument points to IDocument; we need the variable pointed to IMxDocument.
Switch interfaces to IMxDocument

- IDocument's properties are read only
- IMxDocument has write properties and can get neighbors
  - Declare a variable for IMxDocument

Dim pMxDoc As IMxDocument

To get there, declare a variable as IMxDocument, and this is that QueryInterface technique to switch interfaces on the same object.
Switch interfaces to IMxDocument

- IDocument’s properties are read only
- IMxDocument has write properties and can get neighbors
  - Declare a variable for IMxDocument
  - Set the variable equal to the existing variable

Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

Set the variable—the new variable—equal to the variable you’ve already got, ThisDocument.

Now I’ve got two variables to work with in my current map document. I’m going to use this one here to get the selected layer.
Get a layer

- IMxDocument's SelectedLayer property
  - Returns the ILayer interface of the selected layer

Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

On the diagram, we saw the SelectedLayer property returns the ILayer interface of the selected layer.
Get a layer

- IMxDocument's SelectedLayer property
  - Returns the ILayer interface of the selected layer
- Declare a variable to that returned interface

Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

Dim pLayer As ILayer

To get to a neighboring object, you need to declare a variable to the neighboring object. You know which type to put here, or which interface to put here, when you look to the right of the property, ILayer.
Get a layer

- IMxDocument's SelectedLayer property
  - Returns the ILayer interface of the selected layer
- Declare a variable to that returned interface
- Set the variable using Object.Property syntax

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

Dim pLayer As ILayer
Set pLayer = pMxDoc.SelectedLayer
```

To get to the layer, then, run the SelectedLayer property, and that returns the ILayer interface to this variable. So now pLayer points to ILayer.
Set a layer’s Name property

- Check the property’s type
  - Property: Type
  - Name property needs a string

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

Dim pLayer As ILayer
Set pLayer = pMxDoc.SelectedLayer
```

The ILayer interface has the Name property. We want to change the name of the layer. In order to change the name, I’m going to need a string. To the right of Name, you see String.
Set a layer’s Name property

- Check the property’s type
  - Property: Type
    - Name property needs a string
- Set property using Object.Property syntax

```vba
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument

Dim pLayer As ILayer
Set pLayer = pMxDoc.SelectedLayer

pLayer.Name = "Rivers"
```

So here I can easily change the name of the river layer to Rivers by putting a text string in quotes.

Up to this point, I’ve worked with the Layer object and we’ve changed the name of the Layer object.

If we ran the code, it would look like this [opens ArcMap]. The layer name would not change. That’s because we’ve only changed the Layer object in active memory. There are other objects in active memory, like the ArcMap table of contents window. We did not tell the table of contents window to do anything. We need to tell this window to check all the objects it contains and read their current status and refresh with all that current status. So we’ve got another line of code to write.
One more thing...

- Objects live in active memory
- Setting the layer name changes the layer object
- The ArcMap Table of Contents is also an object
  - Have to tell it to refresh with current memory settings
  - Use IMxDocument's UpdateContents method

```vba
pMxDoc.UpdateContents
```

Also, on the IMxDocument interface is the UpdateContents method. This causes the table of contents to refresh. It rereads all the objects it contains—all their properties—and then refreshes itself with those properties.

Alright [opens Microsoft Visual Basic editor]. So, I’ve got that code written here in a subroutine called Change Layer Name. I switched interfaces, I got the layer with the SelectedLayer method, I changed the rivers name, and I did the update contents—refresh the table of contents window. Something that’s important here is the SelectedLayer property. This requires that a layer is selected in the table of contents. Let me bring up ArcMap. Right now, no layers are selected in the table of contents. If we ran that code, we’d get an error message. I need to select a layer in order for that code to run. I’ve selected a layer. Now I’m going to go to the Tools > Macros choice, and I’m going to run the Change Layer Name subroutine, and—look at that—the layer name now changes to Rivers. In that code, we updated the River object—or the Layer object with the name River, and we also updated the table of contents and caused it to refresh so we can now see the layer name.
Well, to kind of wrap things up here, we’ve talked about programming with ArcObjects and using the object model diagrams. The way to do that is to find a property or method on the diagram—part of an interface or a class; that’s your destination. You want to run some property or method there.
Then you need to find a starting point. If that particular object wasn’t a CoClass, you may have to go somewhere else in the diagram to find a starting point. You might start with Application. You might start with ThisDocument, depending upon what’s appropriate. Or, you might start with a CoClass and write two lines of code to create an object out of that CoClass. Once you’ve defined your starting point and your ending point, you really just connect the lines in between.
Review: ArcObjects programming

- Locate a desired class, interface, property, or method
- Pick a starting point: Application, ThisDocument, or CoClass
- Navigate from one object to another, using QI as needed

And you go from one object to the next, declaring variables for the neighboring object, setting them using a property or method on the object you already have.
Well, in the end here, we talked about interface programming, and that all the ArcObjects classes are COM classes; they have interfaces. When you declare your variables, you use those interfaces. Sometimes you have to switch interfaces. If you have an object and you want to get to a lot of its properties and methods, you may have to go from one interface to another using the QueryInterface technique.

Well, we’ve a few questions in the queue. I’m going to turn it over to Jeremiah to answer those questions.

(Jeremiah) Okay. Thanks, Rob.

Okay, Jen from Oklahoma City asked, “What’s the difference between an interface and the object itself? When should you declare an interface and when should you declare the object itself?”

Well, when working with COM objects, we’ll always want to declare an object as an interface. As for the first question, what’s the difference between an interface and an object, an interface is just a piece of an object. So, I like to think of it as maybe looking at a car. You have a car, which is an entire object. On that car there are many ways of getting in that car. There’s through the
trunk, so that might be one interface to get to the engine. There’s going to be another interface to get into the interior through the door. There might be another interface for the trunk. So interfaces are just different ways at getting at the object itself. So, when working with ArcObjects and these COM objects, you’ll always want to declare an object as the interface itself.

Bruce asked, “Are there any plans to provide a .NET interface?”

In our upcoming releases, there won’t be any changes to ArcGIS Desktop itself. We’ll still be using VBA, because it suits the needs of many of our users and it works just great. However, you can access ArcObjects through .NET; it’s just another way that you can work with ArcObjects. So, again, most people will be using ArcObjects through VBA, but you can also use it in other standalone programming languages. And you can use some of the .NET programming languages as well, such as VB .NET.

A few of you have asked this question. I’m going to read this one from Luke. It says, “You can use C++, VBA, VB to develop with ArcObjects. Can I develop with JAVA?”

Right now, at ArcGIS, we are limited to working with COM languages within the Windows environment. In our future release, ArcGIS 9, there are going to be new products that allow us to work in additional languages as well as cross-platform, including JAVA. So we’re going to have a product called ArcGIS Engine Developer Kit and another one called ArcGIS Server, which you’ll be hearing more about.

And I have another related question here. This is from Tom in Billings and it says, “I’ve heard about ArcEngine. Is this related to ArcObjects?”

Yes. This is another product that will be coming out in ArcGIS 9, ArcGIS Engine Developer Kit. What this will allow you to do is create your own standalone applications using ArcObjects rather than customizing the environment. But, again, most of what we’ve been going over today is using VBA to integrate with ArcMap, ArcCatalog, just to customize our environment that is already there.

I'll turn it back over to Rob.
(Rob) Alright, thanks, Jeremiah. There’s one more question here that I’d like to answer. It says, “How do you know which object model diagram to use?”

Well, a lot of times, when you’re programming with ArcObjects, you might be able to figure out the diagram because of the objects you’re working with. If you’re working with the ArcMap objects, the ArcMap diagram is pretty obvious. If you’re working with points and lines and polygons, you might be able to figure out that the Geometry diagram is the one to use. Another place to go, though, is on the ArcObjects Developer Help, one of the diagrams you can open—one of the PDF files you can open—is called All Object Model Diagrams. When you open this one, you see a listing of every single object model diagram. So here, if you know the class or you know the property or method or you have some idea of what it might be called, you might be able to use the search here—in the PDF—to find that property or method or class or interface. It’ll move right to that interface and highlight it for you in the display area. So if you’re not sure which classes to use, open this diagram, type in the name of the class, and it’ll find it for you.
Alright. Those of you who would like to learn a little bit more about ArcObjects, there are a few resources available. One place you might go is take one of the instructor-led training classes. We do have a five-day class; it’s an introductory-level class, and takes you from not knowing much about programming to knowing quite a bit about ArcObjects programming. Then there’s a three-day advanced class. In the advanced class, you’ll learn how to do things like, remember earlier I showed you how I made that extra ContentsView tab, the Catalog tab. You’d learn how to do that in the advanced ArcObjects class.

There are also some Virtual Campus workshops. We have a series of these VBA workshops; there are four of them already. And there’s also a six-module course, *Introduction to Visual Basic 6*.

There are a few books available from ESRI Press; one of them is called *Getting to Know ArcObjects*. This is a 20-chapter, 40-exercise workbook. This is for introductory-level people who don’t know a lot about programming, so the first few chapters teach you about VBA and programming with If Then statements and that type of thing, and then it works slowly through object model diagrams and working with interfaces, and the last 10 or so chapters work with...
about 10 or 12 different object model diagrams to get you used to using some of the main ArcObjects and programming with them.

There are a few online resources available. ArcObjects Online is almost an extension of your help system. It describes how to use properties and methods, and you can look up different classes there and interfaces. There are also white papers and sample code available. Then there’s the VBA Toolkit CD. You can go to the ESRI Web site and request one of these; they are free and they outline, mainly, VBA and how to get going with VBA, but the CD also includes several white papers. It includes the first two chapters of *Exploring ArcObjects*. So the VBA Toolkit CD is—it’s a free place to start. You might request one of these CDs from the Web site.
Alright, it looks like we’re out of time here for today. On behalf of ESRI, I’d like to thank you all for attending.